and thus such scheme provides a control over to RSC 10 to announce only a subset of these channels to IMCs 50 via RASs 30. However, if some stations do not want to encrypt their contents and session announcements at all, this security model should effectively prevent IMCs 50, as well as the nonpaying RASs 30, from receiving the broadcast from those designated stations. Thus, each RSC 10 should maintain a secret key, and encrypt all outgoing content so that only a ciphertext stream is transmitted. In particular, the concept is to generate a symmetric encryption key at RSC 10, and securely distribute this key to a particular RAS 30 upon payment of the required fee. There are many ways this key can be distributed to the local stations, as is known to those having ordinary skill in the art.

The global multicast stream encryption can be extended to RAS 30 as a second level hierarchy. Some of the pay-per-listen and/or pay-per-view programs can be announced to the local multi-cast addresses in any domain using the encryption key so that an appropriate fee collection procedure can be established for the IMCs 50. Any type of encryption can be applied to the audit data of RAS 30, so as to preserve the sensitive information such as the secret keys of RSCs 10, the information for the pay-per-listen and/or pay-per-view channels, the user accounts, and the payment data. The advertising entities can be authenticated so that unauthorized companies could not gain access to AMA 40.

In practice, each RAS 30 generates its own Public Key/Private Key pair. Each RSC 10 generates an SEK key, and begins transmitting the encrypted audio and/or video content. This SEK key should be distributed to the participating RASs 30 in a secure way so that other RASs 30 (which did not pay) cannot obtain this key. As such, the Public Key technology is employed for this purpose. RAS 30 submits the Public Key to RSC 10 along with its payment. Then, RAS 30 receives an Integer ID from RSC 10 which is later used to index the SEK distribution list. RSC 10 collects the Public Keys from RAS 30 and adds these keys to its SEK distribution list upon their payment.

h. Logging Mechanism

One of the purposes of providing a logging mechanism for the system and method of the present inventions is to provide a process for the advertisers to determine when and on which channels to place their advertisements so as to maximize their returns on investment. RTCP is well suited for allowing RAS 30 to collect user-specific and channel-specific listening information. In one embodiment, RAS 30 constantly monitors the number of users receiving the broadcast on each channel, as well as the type of content being transmitted (i.e., the advertisements as opposed to the real content). This is especially advantageous for payment purposes by the advertiser to the local station when the users join or leave the local multicast group at the time when the advertisement starts/stops playing. When RAS 30 detects an "audience change " (i.e., a change in the number of listeners or the type of content), it encapsulates this information into a particular structure, and passes it to the separate logging thread for storing into the log files. The logging thread in turn, buffers this information and periodically writes out the contents of the buffer, in a binary format, to the log files (using a java serialization).

The above-described features - i.e., separate logging thread, output buffering, and binary (as opposed to text) logs - enable a quick output to avoid an interference with the quality of the audio and/or video transmission at run-time. In addition, these features allow for a good scalability as the number of the receivers of the broadcasts increase. A report generation tool can also be used to inspect the log files, and generate the statistics in a format that can be presented to the user. Such tool may support various commands, such as line options that control the way the tool interprets the log and presents its contents to the user.

C. NON-MULTICAST ENABLED NETWORK

The multicasting environment can be implemented for all segments within the system and method of the present invention. Although it may be simpler to implement the multicast communication in the Intranet or within an autonomous system (e.g., a separate domain), in the past the multicast support between the autonomous systems has not been readily available. To extend the above-described

functionality to a network where the multicast communication is not supported, it may be preferable to provide another embodiment of the system and method according to the present invention which is based on the user level or the network level application level.

I. Multicast Tunneling - Network Layer Solution

If there is a lack of the multicast connectivity between some portions of the network, the multicast connectivity can still be used by establishing a multicast tunnel between two different networks using the edge routers. In order to establish the multicast tunnel, it is preferable to provide at least one server running a multicast routing daemon in each such network. However, since this approach is a network layer solution, it may also be preferable if some of the hosts would be running the multicast routing daemon. This approach may also assume that there is a mutual understanding (i.e., interconnectivity) between several connectivity providers.

II. UDP servers - User Level Solution

It may be easier to implement the multicast communication within the Intranet. Since the multicast communication is not yet widely deployed over the wide area network (i.e., some of the intermediate routers may not support the multicast communication), the multicast connectivity between portions of these autonomous systems may not currently exist. Figure 10 shows an exemplary configuration where there is no multicast connectivity in the wide area network, but the multicast communication is enabled within the local area. Thus, there may be islands of multicast enabled networks, but there may not be any multicast connectivity between the islands. In this exemplary configuration, a UDP Server 550 is provided. This UDP Server 550 is co-located with RAS 30. It is also possible that its functionality is provided in the Management Server 200. This UDP server 550 can use a modified version of "rtptrans" which allows a conversion of the audio and/or video stream from the multicast network type to the unicast network type, and vice-versa. "rtptrans" is a conventional RTP translator application which copies RTP packets from any number of unicast and multicast addresses. Alternatively these servers can use "UDP Multicast

Tunneling Protocol" (UMTP) which can set up a connection between the multicast enabled islands by tunneling multicast UDP datagrams inside unicast UDP datagrams.

In particular, whenever any radio station wants to announce its program to the Internet, or if any system wants to broadcast content to the Internet, these devices register with the nearest antenna server. This can be done as described above, where each broadcaster will send its announcement on a specific multicast address in the local domain, and RAS 30 will receive the announcement through SAP. Each RAS 30 is responsible for maintaining a database of the program profile of the set of radio stations announcing in the same domain. If there is no multicast connectivity between antenna servers over the wide area network, then RASs 30 would update each other's program schedules which can be categorized according to, e.g., the News type. Thus, at any point in time, each RAS 30 will have the program schedule of all RSCs 10 broadcasting globally, in addition to its own local program if RAS 30 is broadcasting the local program.

III. Utilization of UDP Multicast Tunneling Protocol (UMTP)

In order to extend this multicast connectivity to all the autonomous systems, LTDP servers that execute the convention UDP Multicast Tunneling Protocol can be implemented for a use with the system and method according to the present invention. For example, the UDP Multicast Tunneling Protocol establishes a connection between two end-UDP servers by tunneling the multicast UDP datagrams of the respective domain inside unicast UDP datagrams. Each UDP server is located within the autonomous system and that autonomous system is multicast capable. In this case, both the end points of the tunnel act as masters. Whenever a tunnel endpoint - whether a master or slave - receives a multicast UDP datagram addressed to a, e.g., group, port, etc., that is currently being tunneled, it encapsulates this datagram, and sends it as a unicast datagram to the other end of the tunnel. Conversely, whenever a tunnel end-point receives, over the tunnel, an encapsulated multicast datagram for a group or port of interest, it decapsulates it and resends it as the multicast datagram.

Each UDP server in an autonomous domain listens to the local common multicast address to find out the announcement within its domain, and passes

it to the other UDP servers in other domain within an encapsulation. Another UDP server, after receiving the local multicast address, decapsulates and announces it on the local multicast address in the other domain. These UDP servers keep listening to each other periodically to update the announcement status. Thus, at any particular point in time, each client knows the program status of several programs that are playing within various domain. If a client prefers to listen to a particular program playing in a different domain, it makes a request on the local common announcement bus. The local UDP server receives the request, and passes this request to the corresponding UDP server in the proper domain. The remote UDP server sends the encapsulated stream on a unicast address, and the local UDP server sends it on the appropriate multicast address in the local domain. It is preferable if there is no overlapping of the multicast addresses with those provided in a different zone (e.g., a zone provided remote from the zone which provide the information on the multicast channel).

IV. RTP Trans

It is also possible to provide an RTPTrans server which converts the multicast stream to the unicast stream, and vice versa. A dedicated RTPTrans server can be provided in each area. Alternatively, the RTPTrans server can be a part of RAS 30. For example, the RSCs 10 in the local area transmit programs to the specified multicast addresses, and send their announcements to the common multicast address. The local RTPTrans server listens to these announcements, and send them to other RTPTrans servers located in different areas, where the announcements are transmitted to the common multicast announcement address in that specific area. Thus, when RAS 30 listens to the common multicast address using SAP, it can obtain the program listing of RSCs 10 in other areas, in addition to the listing in its own area. The RTPtrans server receives the unicast audio and/or video stream from each RSC 10 via other RTPtrans server, and multicasts it on a specific multicast address corresponding to the remote radio station.

D. MOBILITY MANAGEMENT

Another embodiment of the system and method according to the present invention provides a Mobility Management (MM) technique. This technique is especially preferable when IMCs 50 are mobile and wireless. Thus, e.g., area 1 may be covered by one RAS 30 provided one subnet, and area 2 may be covered by a second RAS 30 provided on another subnet. As the mobile and wireless IMC 50 moves from area 1 to area 2, it is essential for the mobile IMC 50 to continue receiving (i.e., without significant interruptions) the broadcast it was receiving from RAS 30 covering area 1. As shall be described in further detail below, one way to accomplish such uninterrupted broadcast is to imitate the streaming of the broadcast that the mobile IMC 50 has been receiving in area 1 into area 2. RAS 30 in area 2 should have adequate information about the mobile IMC 50 traveling into area 2 so that it can now begin streaming with virtually no perceived discontinuity in the broadcast to the mobile IMC 50. Provided below is a description of possible approaches to address the triggering of the multicast streaming in the wireless environment when the mobile IMC 50 moves from one subnet to another.

As described above, each RAS 30 may have two interfaces having respective addresses, one can be a global address and other maybe a local address. It is also possible to utilize one interface for both address configurations. As shown in Figures 11A and 11B, symbols Ia, Ib, Ic represent globally known subnets (e.g., globally addressable subnets) connected to one of the interfaces of respective RAS 30, while symbols ia, ib, ic represent the local subnets (or cells) connected to the secondary interfaces of RAS 30, and could be local to that particular area..

In operation, RAS 30 receives the multicast stream through its global interface and redirects it out through the local interface for IMC 50 in each cell. In the exemplary implementation shown in Figures 11A and 11B, symbols S1, S2 ... S5 represent servers (or RASs 30) which are connected to upstream routers. Each server (with the exception of S2 and S3 which are connected to the same subnet via a multicast switch) is connected to a different subnet, and to a separate interface. Each server is assigned to one particular cellular region, which can be a part of a private subnet dedicated for the local user. The base stations are not shown in Figures 11A

and 11B for the sake of simplicity of the depiction. These base stations can be IP based. It is also possible for the servers to behave as the base stations on one of its local interfaces (e.g. having dual interfaces). It is also possible to connect the second interface of the server to a non-IP based base station (e.g., a layer-2 base station), which would perform the handoff. In the illustrated implementation, the servers S2 and S3 are connected to a multicast switch, which then becomes a part of the same subnet that can manage the traffic using GSMP. GSMP is a General Switch Management Protocol which can be used in multicast transmissions at a switch level. Using GSMP, is possible to save the bandwidth of an adjacent cell if both cells are part of the same subnet. Different exemplary schemes to effectuate the handoff of the broadcast when the mobile IMC 50 moves from area 1 to area 2 (i.e., from subnet S3 to subnet S4) shall be described in further detail below.

I. Post-Registration

The post-registration approach is the easiest approach. However, it may take a long time for the same multicast stream to be directed in the new cell. In an exemplary scenario, the mobile IMC 50 move to a new cell (i.e. from cell ib to cell ic), obtains the new IP address if it is moving to a new subnet, and then sends the join query via RTCP or IGMP scheme. In this case, there may still be a latency during hand-off. This latency can be avoided by other schemes described below.

Popularity based spectrum management to address the limits of spectrum, e.g., a control mechanism to manage an audio/video stream based on a popularity of the program.

In an implementation of an exemplary embodiment of the system and method according to the present invention, the mobile IMC 50 moves to an adjacent cell (i.e., from cell ib to ic), obtains a new IP address via a multicast address dispenser server if it is moving to a new subnet, and sends a "join" message via RTCP or IGMP scheme. After the handover, the mobile IMC 50 would continue to receive the multicast streaming content in the new subnet if there are other active participants in that adjacent cell receiving the content which the mobile IMC 50 wants to receive. If there is no participants which receive a particular streaming content in the adjacent

subnet into which the mobile IMC 50 moves into, then the mobile IMC 50 joins the group by itself after receiving the query from the "first-hop" router (e.g., the router to which the mobile IMC 50 is directly connected to).

It takes some time for the mobile IMC 50 to configure itself after the move, and then join the group. For example, the mobile IMC 50 may wait for 70-75 seconds to receive the multicast traffic it was previously receiving after a handover. It is also possible to use a discovery agent to discover that the mobile IMC 50 has moved to another subnet (i.e., the mobile IMC 50 received a new address). This determination may triggers the above-described joining scenario.

Advantageously, the above-described handover timing can be reduced by exploiting a fast reconfiguration and join time using RTCP (via application layer triggering). For example, if the adjacent cell is not a new subnet , then the mobile IMC 50 does not need to be reconfigured. Indeed, the mobile IMC 50 retains its IP address, and the triggering procedure can still be activated using RTCP by utilizing a variation of GSMP. Otherwise, the streaming content would already be flowing in the adjacent cell via the multicast communication technique.

II. Pre-Registration

Each station (e.g., the servers S1, S2 ... S5) can have multiple neighboring stations (i.e., also servers). For each of these station being shared with another station, it is preferable to issue a multicast announcement (e.g., a multicast address), where each station can determine the program subscribed to, e.g., the group address used by the mobile IMC 50. Just before IMC 50 leaves (or decides to leave) the current cell (a determination which could be based on the threshold value of the received signal), this IMC 50 sends an RTCP message to the local server. The local server then announces this RTCP message to the sharing multicast addresses, where the neighboring stations would be listening to in the global space. The neighboring stations (e.g., servers) connect to the multicast address, and verify it with the information in their own database to determine if this stream has already been transmitted (e.g., if the particular group has already been subscribed to). If another client have been listening to the same stream, then nothing is done. If the broadcast is

not being transmitted, then RAS 30 sends an IGMP message to the upstream router, and passes the stream to the local cells using a local multicast address, even before the mobile IMC 50 moves to the new cell. Thus, a soft hand-off is emulated for the associated stream. As soon as the mobile IMC 50 moves to the next cell, it can still receive the same stream without any interruption. The mobile IMC 50 sends an RTCP BYE message to the server as it move away from the previous server.

III. Pre-Registration with Multicast Agent

In this scheme, a multicast agent is utilized to take care of the multicast stream. The multicast agent can be provided within each router, which sends these streams to the respective global multicast addresses (e.g., for the area where these clients are trying to move in) in each subnet for a specific period of time, as determined by a timer associated with the subnet. Thus, each neighboring server receives the stream irrespective of whether the mobile IMC 50 is moving into that cell or not. As soon as the mobile IMC 50 moves into the new cell, it sends an RTCP signal to alert that the mobile IMC 50 has moved in, thus the timer does not need to be triggered.

IV. During Registration

In another scheme according to the present invention, this information can be passed on, as a part of a registration method. When the mobile IMC 50 moves in and attempts to acquire the address in the local subnet, it can send the request for that stream in its DHCP option regarding the address it has been listening to. However, in that case, the server may also be a registration server. Thus, at the time of obtaining the IP address from the DHCP server, the mobile IMC 50 can send the local multicast address to the server, and depending on whether the server is already a part of the multicast tree, it would ignore this request or re-join the tree.

V. Proxy Registration

Another scheme can deploy a proxy agent in each subnet. These proxy agents join the upstream multicast tree on behalf of the servers, even before the

mobile IMC 50 moves into the cell. The neighboring proxy server would then listen to a common multicast address to determine the impending host's subscribed multicast address.

The foregoing describes exemplary embodiments of the present invention. Various modifications and alterations to the described embodiments will be apparent to those skilled in the art in view of the teachings herein. For example, the system and method according to the present invention can also be used for either wired or wireless teleconferencing over the Internet using at least in part the multicast communication technique. It will thus be appreciated that those skilled in the art will be able to devise numerous systems and methods which, although not explicitly shown or described herein, embody the principles of the present invention, and are thus within the spirit and scope of the present invention.

CLAIMS

1. A method for providing a broadcast of content to a receiver via a communication network, comprising the steps of:

a) receiving the broadcast on at least one global multicast channel;

b) associating at least one local multicast channel with the at least one global multicast channel;

c) connecting the receiver to the at least one local multicast channel; and

d) routing the broadcast from the at least one global multicast channel to the at least one local multicast channel to provide the broadcast to the receiver.

2. The method according to claim 1, further comprising the step of:

e) receiving a request from the receiver to receive the broadcast.

3. The method according to claim 1, further comprising the steps of:

f) inserting the broadcast into the at least one global multicast channel; and

g) transmitting the broadcast at the at least one global multicast channel from a global server to a local server.

4. The method according to claim 3,

wherein the at least one global multicast channel is a plurality of global multicast channels, and the at least one local multicast channel is a plurality of local multicast channels,

wherein the broadcast is inserted into a first global channel of the global multicast channels,

wherein the first global channel is associated with a first local channel of the local multicast channels, and

wherein the receiver receives the broadcast from the first global channel on the first local channel.

5. The method according to claim 4, wherein the broadcast is inserted into the first global channel by the global server, and wherein the global multicast channels are received by the local server.

6. The method according to claim 5, further comprising the steps of:

h) at the global server, inserting a further broadcast of content into a second global channel of the global multicast channels;

i) receiving a request from the receiver to receive the further broadcast from the local server;

j) if the second global channel is not available to the local server, obtaining access for the local server to the second global channel;

k) after step (i), associating the second global channel with a second local channel of the local multicast channels; and

l) providing the further broadcast to the receiver by connecting the receiver to the second local channel and routing the further broadcast from the second global channel to the second local channel.

7. The method according to claim 1,

wherein the at least one global multicast channel is a plurality of global multicast channels,

wherein the at least one local multicast channel is a plurality of local multicast channels,

wherein the broadcast is inserted into a particular global channel by a global broadcasting device, and

wherein the broadcast from the global multicast channels are received by a local broadcasting device.

8. The method according to claim 7, further comprising the steps of:

m) inserting a local broadcast into a particular local channel of the local multicast channels, the local broadcast being different from the inserted broadcast; and

n) if the receiver issues a request to receive the local broadcast, establishing a communication link for the receiver to the particular local channel to receive the local broadcast.

9. The method according to claim 1, further comprising the step of:

o) at a predetermined time and using a multicast communication, determining a number of receivers which are receiving the broadcast.

10. The method according to claim 1, wherein the receiver includes an Internet Protocol (IP) interface which enables the receiver to receive the broadcast via an IP-type multicast communication.

11. The method according to claim 1, wherein the receiver is wireless, and receives the broadcast in a first subnet using a multicast communication, and further comprising the steps of:

p) receiving, from the receiver moving from the first subnet to a second subnet, a request to receive the broadcast in the second subnet; and

q) after receiving the request from the receiver, providing the broadcast to the wireless receiver in the second subnet using the multicast communication.

12. The method according to claim 11, further comprising the step of:

r) stopping a transmission of the broadcast in the first subnet after receiving the request from the receiver.

13. The method according to claim 1, wherein normal content of the broadcast has at least one break of respective predetermined duration, and further comprising the steps of:

s) inserting respective predefined content data into the at least one break in the normal content of the broadcast; and

t) providing the broadcast to the receiver after the respective predefined content data is inserted into the at least one break of the normal content of the broadcast.

14. The method according to claim 13, wherein the predefined content includes at least one of an advertisement, a station break announcement, a promotion and other pre-recorded content.

15. The method according to claim 8, wherein the local broadcast has at least one break at a respective time and of a respective duration, and further comprising the steps of:

u) inserting respective predefined content into the local broadcast during the at least one break in the normal content of the local broadcast; and

v) providing the local broadcast to the receiver after the respective predefined content of the local broadcast is inserted into the at least one break of the normal content of the local broadcast.

16. The method according to claim 13, wherein the particular data includes at least one of an advertisement, a station break announcement, a promotion and pre-recorded content for global broadcast.

17. A method for providing a respective predefined content to a receiver during a real-time broadcast of normal content, comprising the steps of:

- receiving the real-time broadcast of normal content from a remote device via a multicast communication, the real-time broadcast including information indicative of a respective time and a duration of at least one break in the broadcast of the normal content;

- inserting the respective predefined content into the real-time broadcast during the at least one break in the normal content; and

- providing the real-time broadcast to the receiver after the respective predefined content have been inserted into the at least one break in the normal content of the real-time broadcast.

18. The method according to claim 17, wherein the respective predefined content includes at least one of an advertisement, a station break announcement, a promotion and other pre-recorded content for global broadcast.

19. The method according to claim 17, wherein the real-time broadcast is received on at least one global multicast channel, and further comprising the steps of:

- associating at least one local multicast channel with the at least one global multicast channel; and

- establishing a network link between the receiver and the at least one local multicast channel, and wherein the real-time broadcast is provided to the receiver by routing the real-time broadcast from the at least one global multicast channel to the at least one local multicast channel.

20. The method according to claim 17, wherein the receiver is wireless and receives the real-time broadcast in a first subnet using a multicast communication, and further comprising the steps of:

- receiving, from the receiver moving from the first subnet to a second subnet, a request to receive the real-time broadcast in the second subnet; and

- after receiving the request from the receiver, providing the real-time broadcast to the wireless receiver in the second subnet using the multicast communication.

21. The method according to claim 17, wherein the receiver includes an Internet Protocol (IP) interface which enables the receiver to receive the real-time broadcast via an IP-type multicast communication.

22. A method for providing and maintaining a real-time broadcast to a wireless receiver on a communications network, comprising the steps of:

providing the real-time broadcast into the receiver in a first subnet using a multicast communication;

receiving from the wireless receiver, moving from the first subnet to a second subnet, a request to receive the real-time broadcast in the second subnet; and

after receiving the request from the wireless receiver, providing the real-time broadcast to the wireless receiver in the second subnet using the multicast communication.

23. The method according to claim 22, further comprising the step of:

- stopping a transmission of the real-time broadcast in the first subnet after receiving the request from the receiver.

24. The method according to claim 22, wherein the wireless receiver includes an Internet Protocol (IP) interface which enables the receiver to receive the real-time broadcast via an IP-type multicast communication.

25. The method according to claim 22, wherein the real-time broadcast is received on at least one global multicast channel, and further comprising the steps of:

- associating at least one local multicast channel with the at least one global multicast channel; and

- establishing communication to the wireless receiver over the at least one local multicast channel, and wherein the real-time broadcast is provided to the wireless receiver by routing the real-time broadcast from the at least one global multicast channel to the at least one local multicast channel.

26. The method according to claim 22, wherein normal content of the real-time broadcast has at least one break at a respective time and for a respective duration, and further comprising the steps of:

inserting respective predefined content into the real-time broadcast during the at least one break in the normal content; and

providing the real-time broadcast to the wireless receiver after the respective predefined content is inserted into the real-time broadcast during the at least one break in the normal content.

27. A receiver, comprising:

a tuner receiving at least one of a radio broadcast and a television broadcast; and

an Internet Protocol-type communication device configured to receive a real-time Internet Protocol broadcast via a multicast communication, the analog tuner being coupled to the Internet Protocol-type communication device.

28. The receiver according to claim 27, further comprising:

a switching device coupled between the Internet Protocol-type communication device and the tuner, the switching device being switchable between a first state and a second state, the first state enabling the tuner to receive broadcast signals, the second state enabling the Internet Protocol-type communication device to receive Internet Protocol type data using the multicast communication.

29. The receiver according to claim 27, wherein the Internet Protocol-type communication device is connected to at least one local multicast channel for receiving the real-time broadcast from a global multicast channel.

30. The receiver according to claim 27,

wherein the receiver is wireless, and the Internet Protocol-type communication device receives the real-time broadcast in a first subnet using the multicast communication,

wherein, prior to the wireless receiver moving from the first subnet to a second subnet, the Internet Protocol-type communication device transmits a request to receive the real-time broadcast in the second subnet; and

wherein, after transmitting the request, the Internet Protocol-type communication device receives the real-time broadcast in the second subnet by utilizing the multicast communication.

31. A method for monitoring a number of receivers that receive a broadcast via a communication network, comprising the steps of:

providing the broadcast to at least one of the receivers on at least one local multicast channel; and

at a predetermined time and using a multicast communication, explain how the number of the receivers which are receiving the broadcast the number being determined by receiving information from the receivers indicative of the broadcast being received by the receiving.

32. A device for providing a broadcast of content to a receiver via a communication network, comprising the steps of:

a communication device communicating with at least one global multicast channel to receive the broadcast;

at least one local multicast channel; and

a processing device associating the at least one local multicast channel with the at least one global multicast channel, and routing the broadcast from the at least one global multicast channel to the at least one local multicast channel to provide the broadcast to the receiver.

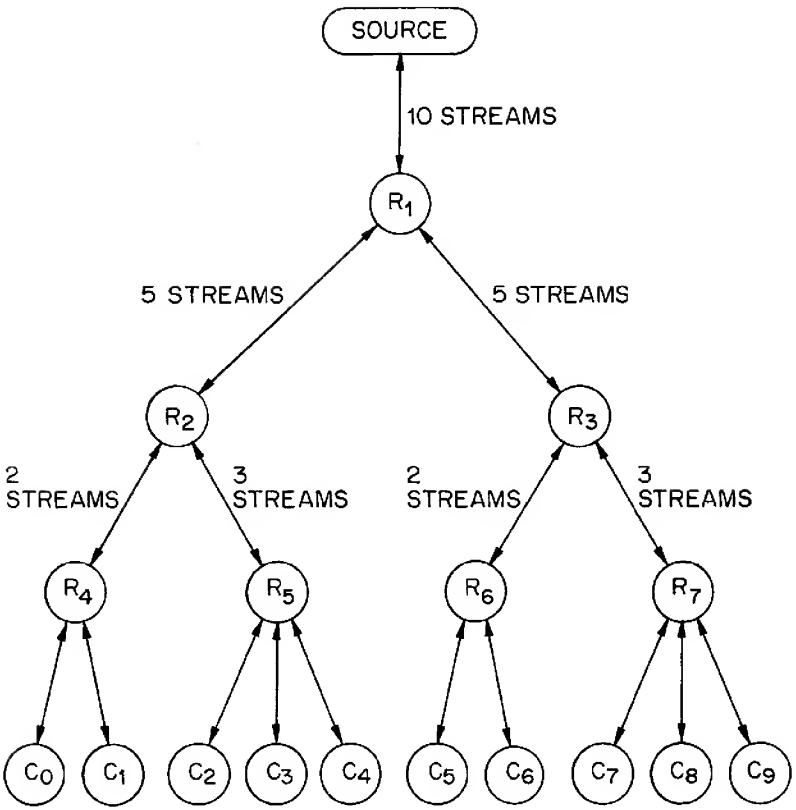


FIG. 1
PRIOR ART

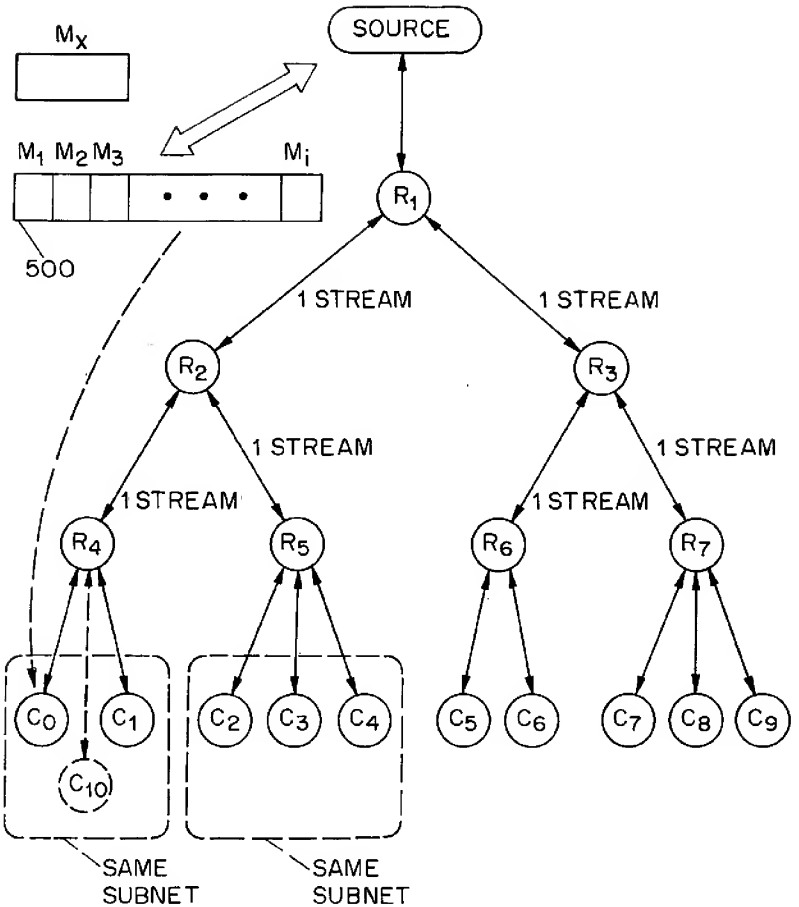

M_X
SOURCE
M_1 M_2 M_3
M_i
R_1
500
1 STREAM
1 STREAM
R_2
R_3
1 STREAM
1 STREAM
1 STREAM
1 STREAM
R_4
R_5
R_6
R_7
C_0
C_1
C_2
C_3
C_4
C_5
C_6
C_7
C_8
C_9
C_{10}
SAME SUBNET
SAME SUBNET

FIG. 2

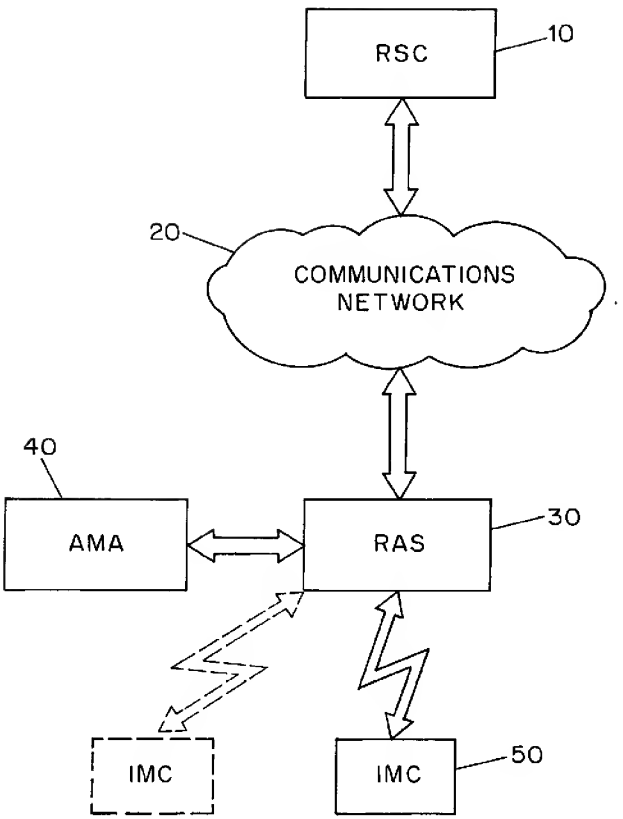


FIG. 3

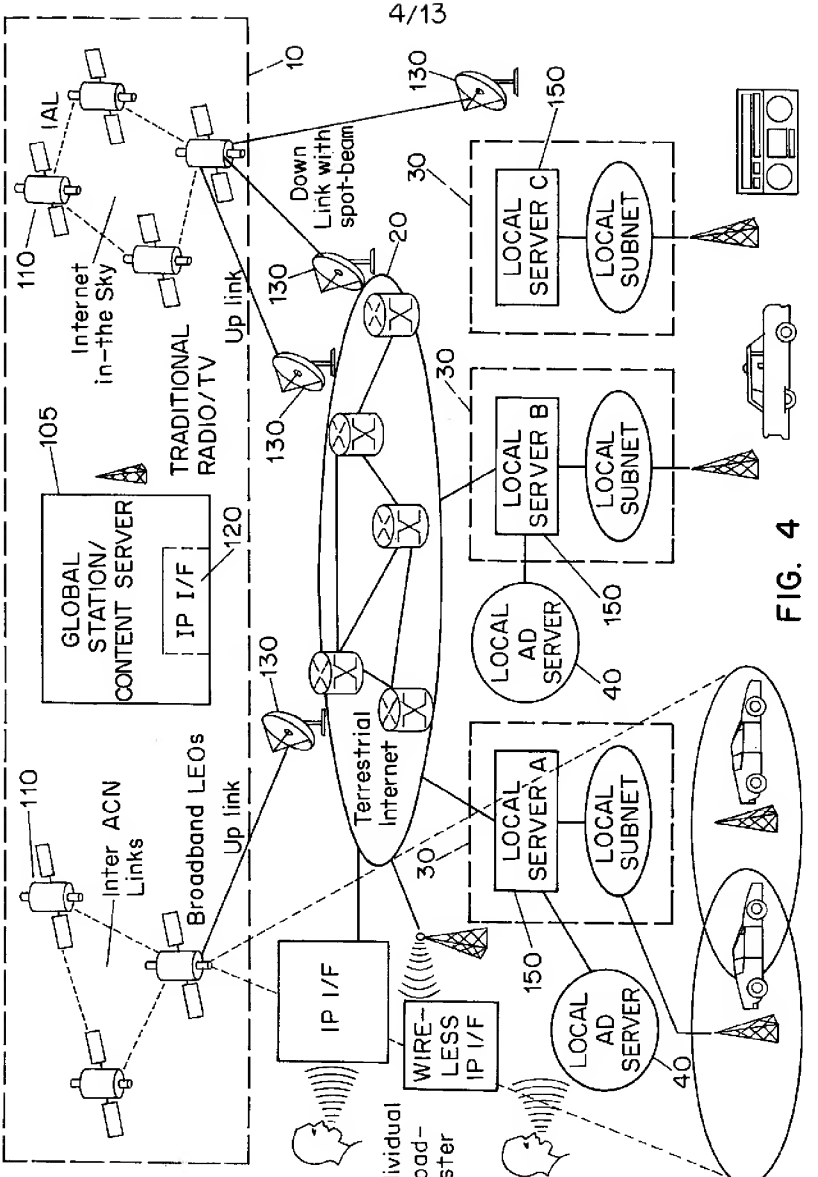

10
IAL
110
Internet in-the Sky
105
GLOBAL STATION/ CONTENT SERVER
IP I/F
120
TRADITIONAL RADIO/TV
Up link
Down Link with spot-beam
130
20
30
LOCAL SERVER C
LOCAL SUBNET
150
LOCAL SERVER B
LOCAL AD SERVER
40
Terrestrial Internet
LOCAL SERVER A
Inter ACN Links
Broadband LEOs
IP I/F
WIRE-LESS IP I/F
Individual Broad-caster

FIG. 4

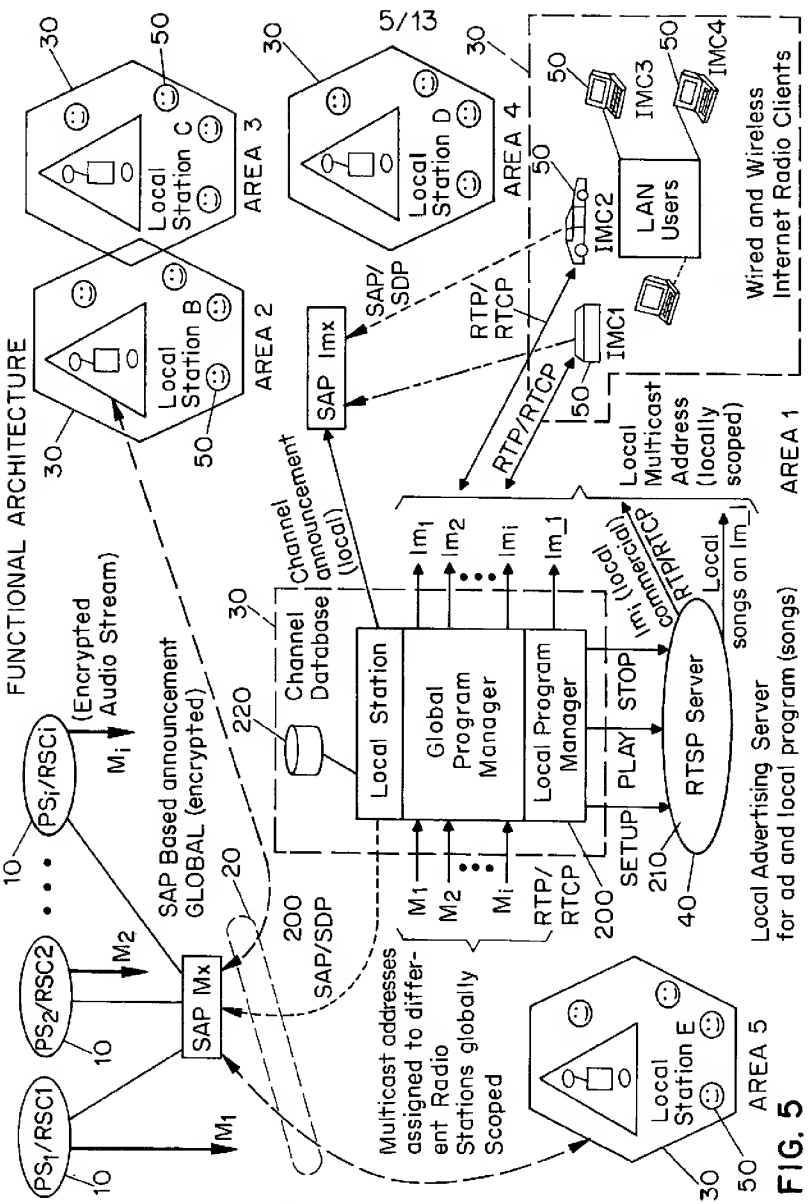

FUNCTIONAL ARCHITECTURE
5/13
PS1/RSC1
PS2/RSC2
PSi/RSCi
(Encrypted Audio Stream)
M1
M2
Mi
SAP Mx
SAP Based announcement GLOBAL (encrypted)
SAP/SDP
Channel Database
Channel announcement (local)
Local Station
Global Program Manager
Local Program Manager
SETUP
PLAY
STOP
RTP/RTCP
RTSP Server
Im1
Im2
Imi
Im_1
Imi (local) commercial
Local songs on Im_1
Local Advertising Server for ad and local program (songs)
SAP Imx
SAP/SDP
RTP/RTCP
Local Multicast Address (locally scoped)
Multicast addresses assigned to different Radio Stations globally Scoped
Local Station B
Local Station C
Local Station D
Local Station E
AREA 1
AREA 2
AREA 3
AREA 4
AREA 5
IMC1
IMC2
IMC3
IMC4
LAN Users
Wired and Wireless Internet Radio Clients

FIG. 5

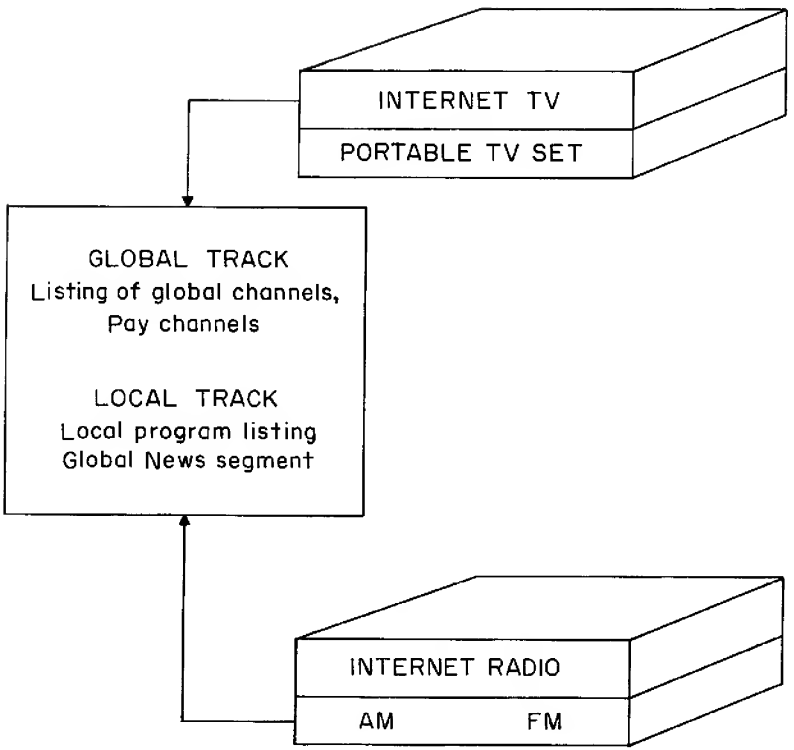

INTERNET TV
PORTABLE TV SET
GLOBAL TRACK
Listing of global channels,
Pay channels
LOCAL TRACK
Local program listing
Global News segment
INTERNET RADIO
AM
FM

FIG. 6A

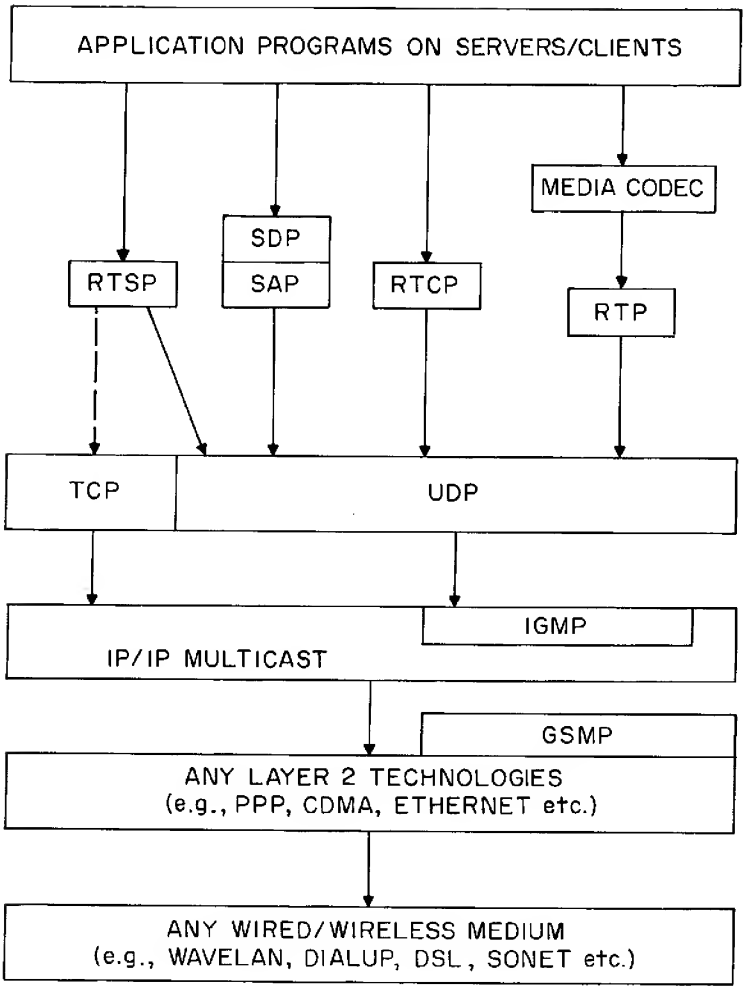

APPLICATION PROGRAMS ON SERVERS/CLIENTS
MEDIA CODEC
SDP
SAP
RTSP
RTCP
RTP
TCP
UDP
IGMP
IP/IP MULTICAST
GSMP
ANY LAYER 2 TECHNOLOGIES
(e.g., PPP, CDMA, ETHERNET etc.)
ANY WIRED/WIRELESS MEDIUM
(e.g., WAVELAN, DIALUP, DSL, SONET etc.)

FIG. 6B

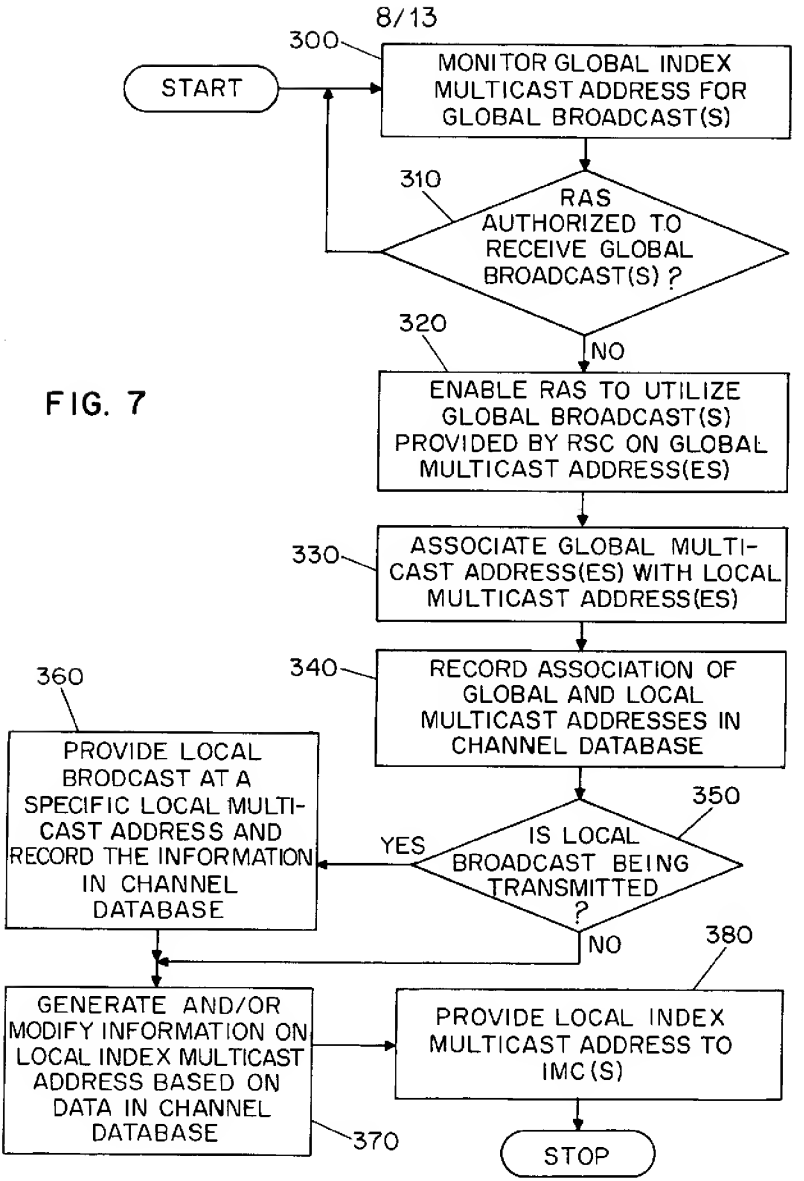

START
300
MONITOR GLOBAL INDEX MULTICAST ADDRESS FOR GLOBAL BROADCAST(S)
310
RAS AUTHORIZED TO RECEIVE GLOBAL BROADCAST(S) ?
NO
320
ENABLE RAS TO UTILIZE GLOBAL BROADCAST(S) PROVIDED BY RSC ON GLOBAL MULTICAST ADDRESS(ES)
330
ASSOCIATE GLOBAL MULTICAST ADDRESS(ES) WITH LOCAL MULTICAST ADDRESS(ES)
340
RECORD ASSOCIATION OF GLOBAL AND LOCAL MULTICAST ADDRESSES IN CHANNEL DATABASE
350
IS LOCAL BROADCAST BEING TRANSMITTED ?
YES
360
PROVIDE LOCAL BRODCAST AT A SPECIFIC LOCAL MULTICAST ADDRESS AND RECORD THE INFORMATION IN CHANNEL DATABASE
NO
380
PROVIDE LOCAL INDEX MULTICAST ADDRESS TO IMC(S)
GENERATE AND/OR MODIFY INFORMATION ON LOCAL INDEX MULTICAST ADDRESS BASED ON DATA IN CHANNEL DATABASE
370
STOP

FIG. 7

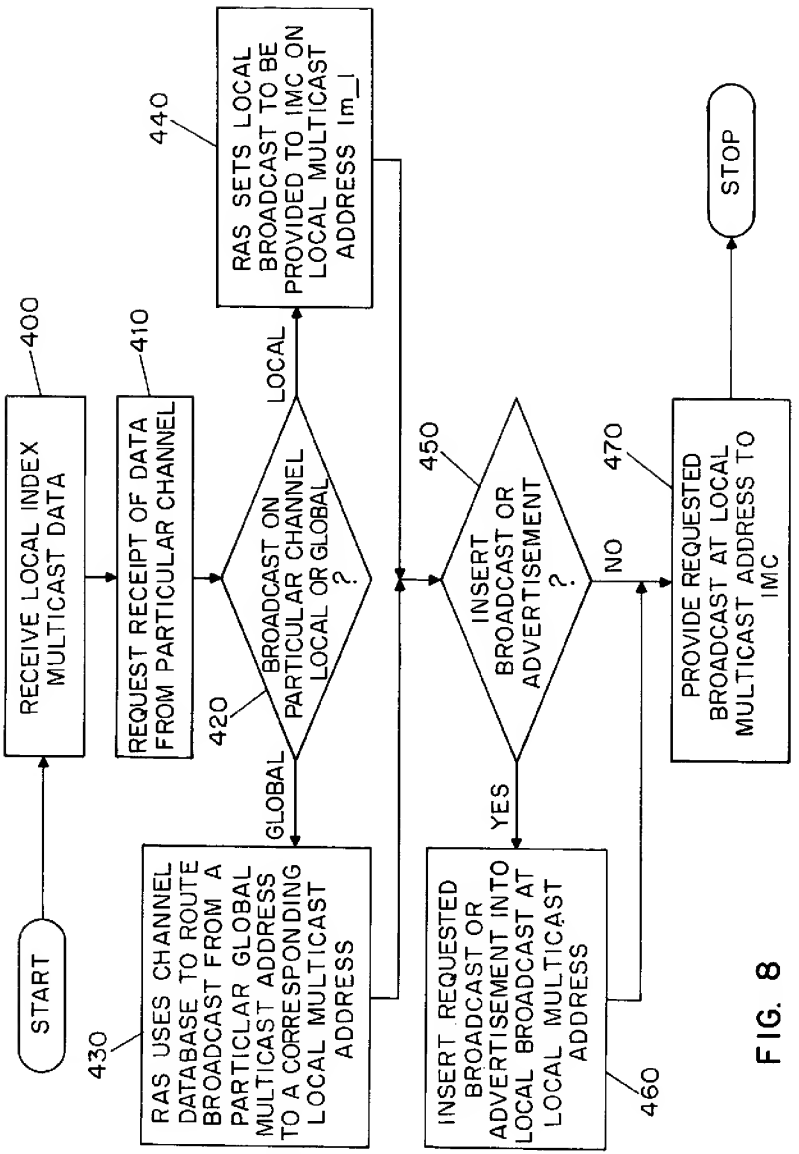
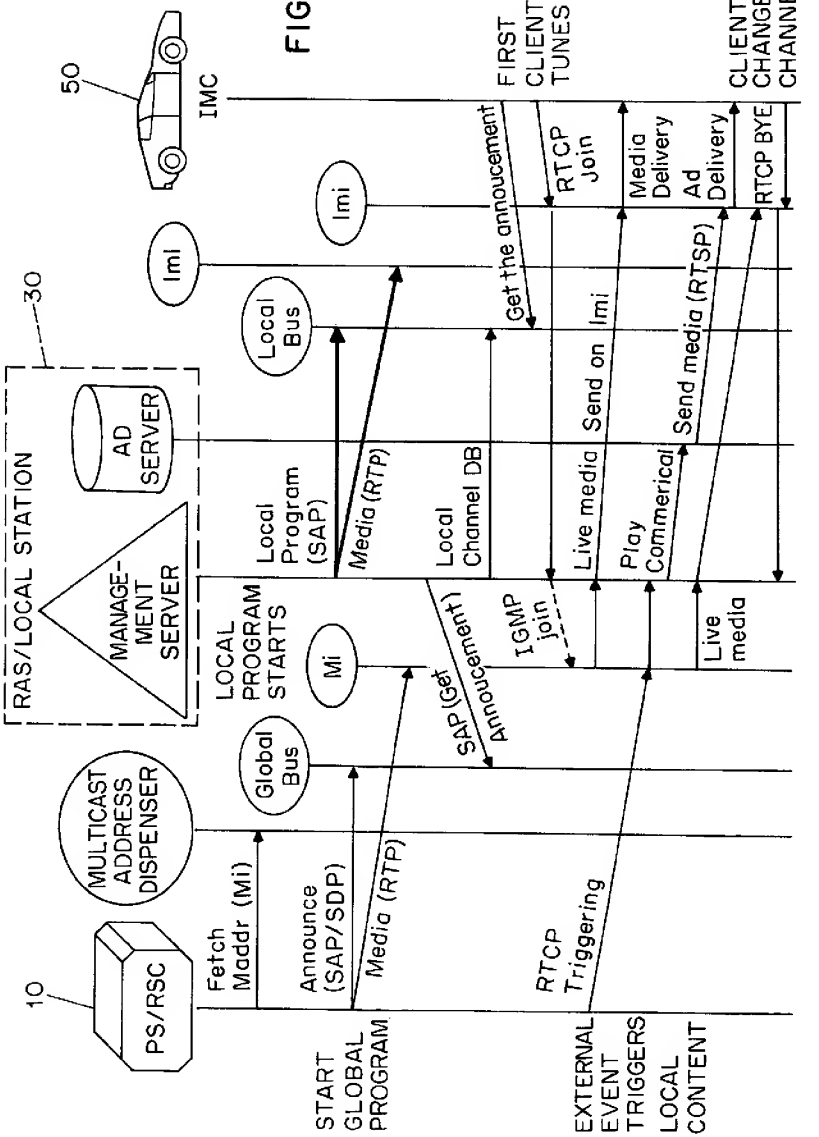

START
RECEIVE LOCAL INDEX MULTICAST DATA
400
REQUEST RECEIPT OF DATA FROM PARTICULAR CHANNEL
410
BROADCAST ON PARTICULAR CHANNEL LOCAL OR GLOBAL ?
420
LOCAL
GLOBAL
RAS SETS LOCAL BROADCAST TO BE PROVIDED TO IMC ON LOCAL MULTICAST ADDRESS Im_l
440
RAS USES CHANNEL DATABASE TO ROUTE BROADCAST FROM A PARTICLAR GLOBAL MULTICAST ADDRESS TO A CORRESPONDING LOCAL MULTICAST ADDRESS
430
INSERT BROADCAST OR ADVERTISEMENT ?
450
YES
NO
INSERT REQUESTED BROADCAST OR ADVERTISEMENT INTO LOCAL BROADCAST AT LOCAL MULTICAST ADDRESS
460
PROVIDE REQUESTED BROADCAST AT LOCAL MULTICAST ADDRESS TO IMC
470
STOP

FIG. 8

FIG. 9


10
PS/RSC
MULTICAST ADDRESS DISPENSER
RAS/LOCAL STATION
30
MANAGE-MENT SERVER
AD SERVER
50
IMC
Global Bus
Local Bus
Imi
Mi
LOCAL PROGRAM STARTS
Local Program (SAP)
Local Channel DB
Fetch Maddr (Mi)
Announce (SAP/SDP)
Media (RTP)
SAP (Get Annoucement)
IGMP join
Get the annoucement
FIRST CLIENT TUNES
RTCP Join
Media Delivery
Ad Delivery
RTCP BYE
CLIENT CHANGES CHANNEL
Live media
Send on Imi
Play Commerical
Send media (RTSP)
RTCP Triggering
START GLOBAL PROGRAM
EXTERNAL EVENT TRIGGERS
LOCAL CONTENT

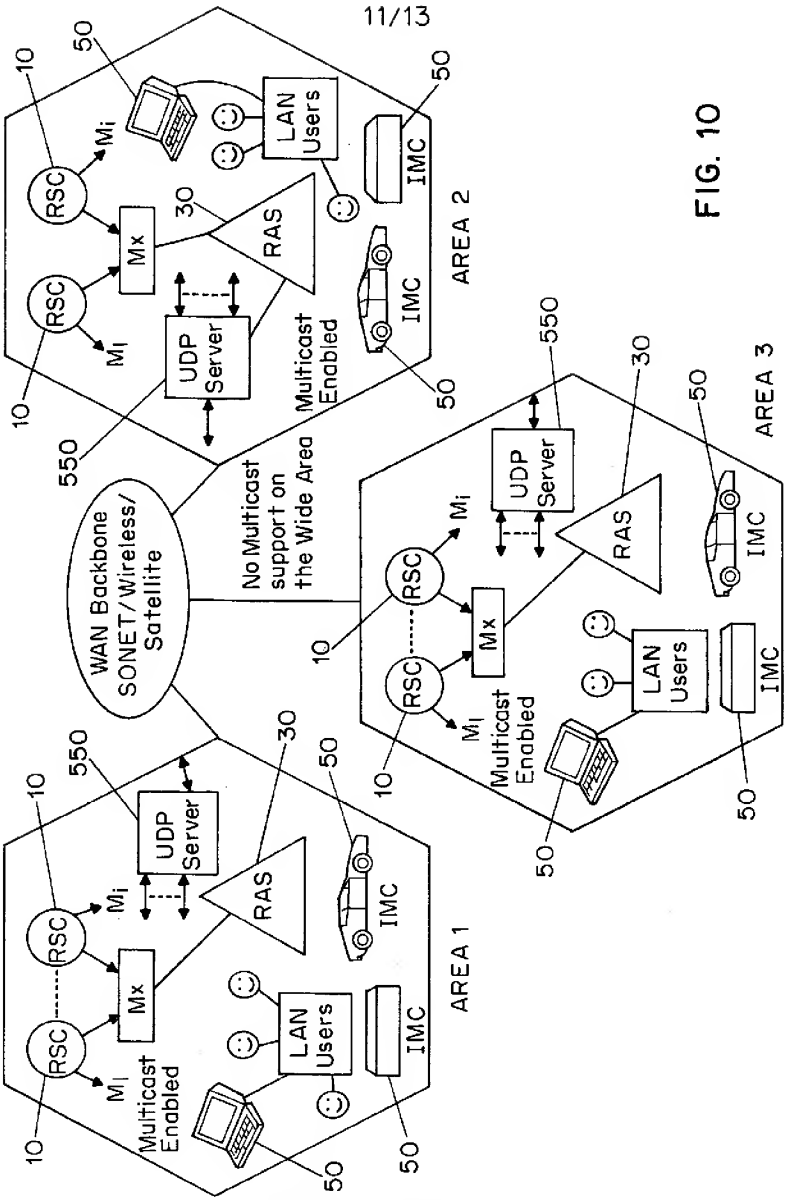

WAN Backbone
SONET/Wireless/
Satellite
No Multicast
support on
the Wide Area
AREA 1
AREA 2
AREA 3
Multicast
Enabled
RSC
Mx
Mi
Ml
UDP
Server
RAS
IMC
LAN
Users
10
30
50
550

FIG. 10

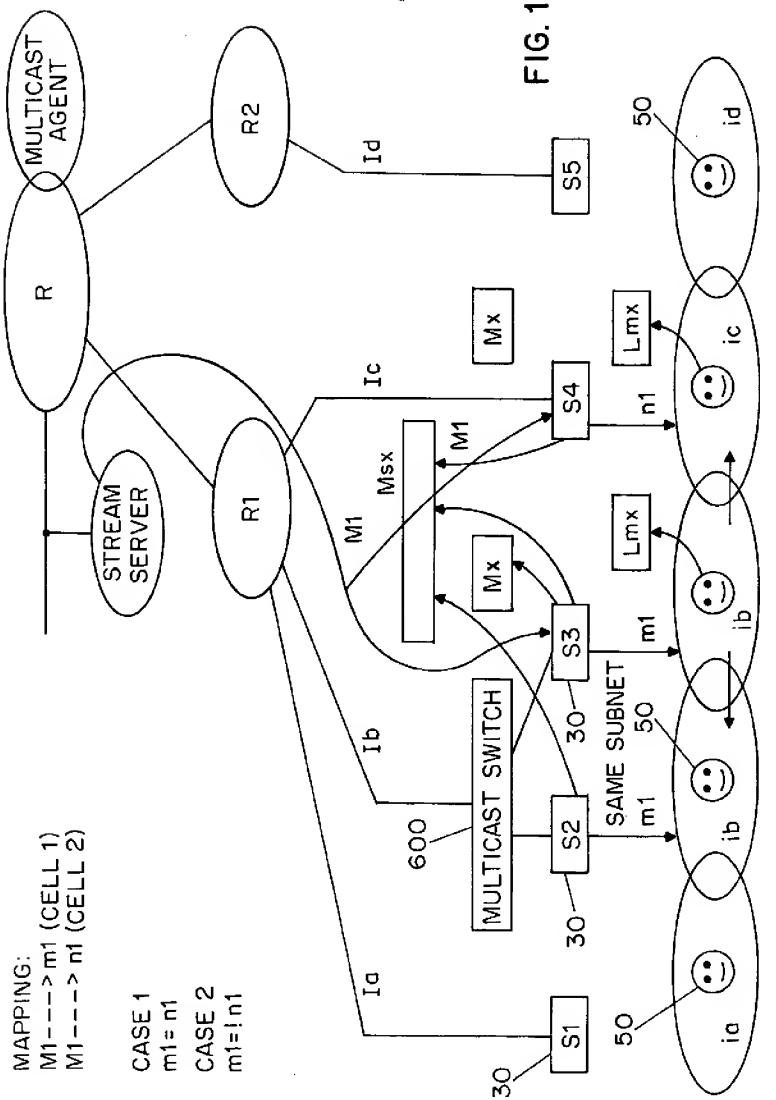

MULTICAST AGENT
R
R2
Id
S5
STREAM SERVER
R1
Ic
Mx
S4
Msx
M1
Lmx
n1
ic
id
Ib
600
MULTICAST SWITCH
S3
S2
S1
30
SAME SUBNET
m1
50
ib
ia
Ia
MAPPING:
M1---> m1 (CELL 1)
M1---> n1 (CELL 2)
CASE 1
m1= n1
CASE 2
m1=! n1

FIG. 11A

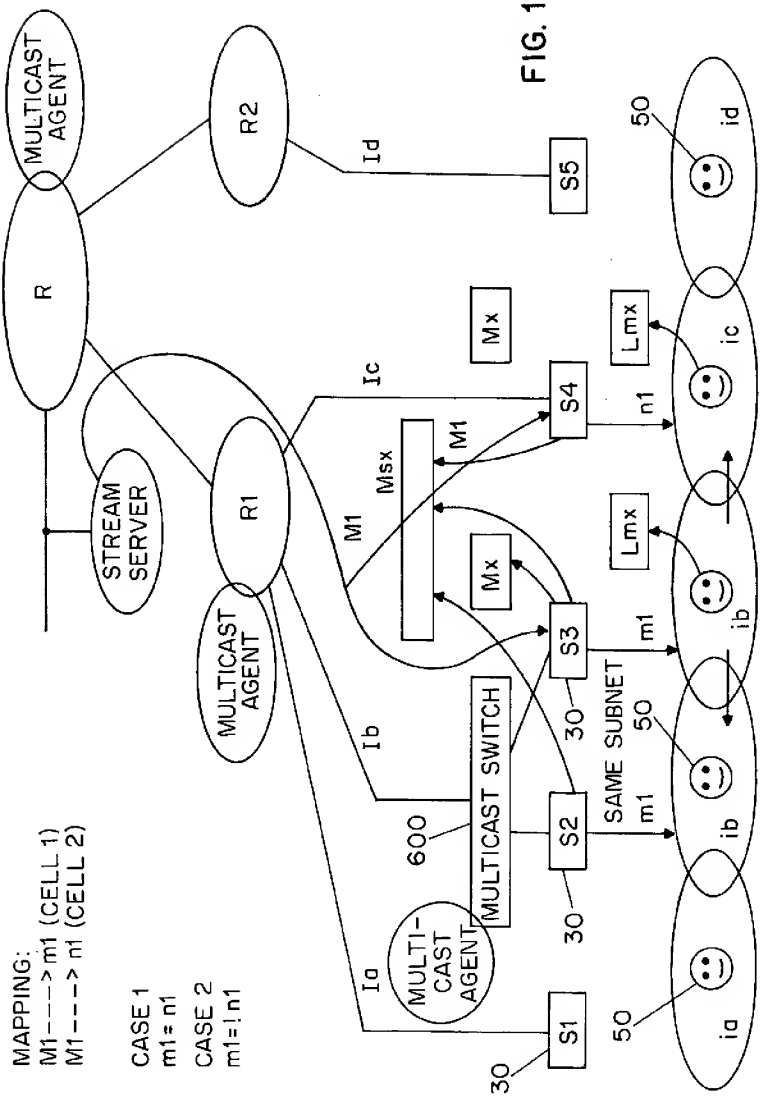

MULTICAST AGENT
R
R2
Id
S5
STREAM SERVER
R1
Ic
Mx
S4
Lmx
n1
M1
Msx
M1
Mx
Lmx
MULTICAST AGENT
Ib
S3
m1
ib
MULTICAST SWITCH
600
30
SAME SUBNET
50
S2
m1
MULTI-CAST AGENT
Ia
30
S1
50
ia
ib
ic
id
50
MAPPING:
M1---->m1 (CELL 1)
M1---->n1 (CELL 2)
CASE 1
m1= n1
CASE 2
m1=! n1

FIG. 11B

RAS.c Thu Jun 17 14:[redacted]:14 1999 1

```
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
#include <net/if.h>
#include <fcntl.h>
#include <stropts.h>
#include <pthread.h>
#include <sched.h>
#include <stdlib.h>
#include <limits.h>
#include "RAS.h"


extern uint32_t random32(int type);

int total_threads = 0;
int current_state_index = -1;
State *state_array;
int server_port, multicast_port;
int connfd;


main(int argc, char **argv)
{
  int listenfd, fd;
  int i, n;
  struct sockaddr_in servaddr, cliaddr;
  int len = sizeof(cliaddr);
  RequestPkt *rp;
  char buffer[sizeof(RequestPkt) + MAX_FILENAME];
  int state_index;

  if (argc < 3) {
        printf("usage: RAS <RAS port> <multicast port>\n");
        exit(1);
  }

  argv++;
  server_port = atoi(*argv);

  argv++;
  multicast_port = atoi(*argv);

  state_array = (State *) malloc (sizeof (State) * MAX_STATES);
  for (i=0; i<MAX_STATES; i++) {
    state_array[i].valid = FALSE;
  }

 listenfd = socket (AF_INET, SOCK_STREAM, 0);
 if (listenfd < 0) {
   perror ("socket");
   exit(1);
 }

 bzero (&servaddr, sizeof (servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servaddr.sin_port = server_port;

 if (bind(listenfd, (struct sockaddr *) &servaddr, sizeof (servaddr)) < 0) {
```

EJ339573277US

```
    perror ("bind");
    exit(1);
  }

  if (listen (listenfd, 1024) < 0) {
    perror ("listen");
    exit(1);
  }

  connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &len);
  if (connfd < 0) {
    perror ("accept");
    exit(1);
  }

  rp = (RequestPkt *) buffer;

  while (TRUE) {
    printf("-------------------------\n");

    if ((n = read(connfd, rp, MAX_REQUEST_LENGTH)) <= 0) {
      perror ("read");

      for(i = 0; i <= current_state_index; i++) {
        if (state_array[i].valid) {
          stop(i);
        }
      }

      exit(1);
    }

//    if (n == 0) continue;

    for(i = 0; i < 12; i++) {
      printf("%x\n", buffer[i]);
    }
    printf("RAS: main: m_addr=%x, lm_addr=%x\n", (uint32_t)rp->m_addr,
           (uint32_t)rp->lm_addr);


    state_index = findLM(rp->lm_addr);

    switch (rp->request_type) {

    case PLAY:
      printf("PLAY\n");
      sendStatus(play(state_index, rp->m_addr, rp->lm_addr));
      break;

    case COMMERCIAL:
      printf("COMM\n");

      printf("state_index=%d, path_length=%d, path=%s\n",
             state_index, rp->path_length, rp->path);

      sendStatus(commercial(state_index, rp->path_length, rp->path));
      break;

    case LOCAL_PLAY:
      printf("LOCAL PLAY\n");

      printf("path_length=%d, path=%s\n",
             rp->path_length, rp->path);
```

```
      sendStatus(localPlay(rp->lm_addr, rp->path_length, rp->path));
      break;

    case STOP:
      printf("STOP\n");
      if (state_index < 0) sendStatus (ERROR);
      else stop (state_index);
      sendStatus(OK);
      break;

    default:
      printf("ERROR\n");
      sendStatus(ERROR);
    }
  }
}

int findNewIndex()
{
  int i;

  printf("findNewIndex(): total_treads=%d, current_state_index=%d\n",
         total_threads, current_state_index);

  if (current_state_index == total_threads - 1) {
    if (current_state_index + 1 >= MAX_STATES) {
      printf("findNewIndex1(): returning -1 \n");
      return -1;
    }
    else {
      printf("findNewIndex2(): returning %d\n", current_state_index + 1);
      return ++current_state_index;
    }
  }
  else
    for (i=0; i<=current_state_index; i++) {
      if (!state_array[i].valid) {
        printf("findNewIndex3(): returning %d\n", i);
        return i;
      }
    }
  printf("findNewIndex(): impossible!!!\n");
}


int findLM (uint32_t lm_addr)
{
  int i;
  for (i=0; i<=current_state_index; i++) {
    if (state_array[i].valid && state_array[i].lm_addr == lm_addr) {
      printf("findLM(): returning index %d\n", i);
      return i;
    }
  }
  printf("findLM(): returning -1\n");
  return -1;
}


int commercial (int state_index, int path_length, const char *path) {
  FILE *fd;

  printf("commercial(state_index=%d, path_length=%d)\n",
```

```
        state_index, path_length);

  if (state_index == -1 || path_length <= 0) return ERROR;
  if (state_array[state_index].fd) fclose(state_array[state_index].fd);

  fd = fopen(path, "r");
  if (fd == NULL) {
    printf("commercial(): error opening file\n");
    return ERROR;
  }
  state_array[state_index].fd = fd;
  state_array[state_index].request_type = COMMERCIAL;
  return OK;
}

int localPlay (uint32_t lm_addr, int path_length, const char *path) {
  FILE *fd;
  int state_index, status;

  printf("localPlay(path_length=%d, path=%s\n",
         path_length, path);

  if (path_length <= 0) return ERROR;

  fd = fopen(path, "r");
  if (fd == NULL) {
    printf("localPlay(): error opening file\n");
    return ERROR;
  }

  state_index = findNewIndex();
  if (state_index == -1) return STATES_FULL;

  /* create new state entry */
  state_array[state_index].fd = fd;
  state_array[state_index].request_type = LOCAL_PLAY;
  state_array[state_index].m_addr = NULL;
  state_array[state_index].lm_addr = lm_addr;
  state_array[state_index].valid = TRUE;

  total_threads++;

  /* create a new playing thread (using old state entry) */
  if ((status =
       pthread_create(&(state_array[state_index].tid),
                      NULL, processRequest, (void *) state_index)) != 0) {
    printf("play2: can't create thread: error # %d\n", status);
    exit(1);
  }
  return OK;
}


void sendStatus (uint8_t status)
{
  uint8_t s = status;

  write (connfd, &s, 1);
}


void *processRequest(void* index)
{
  struct ip_mreq mreq_m, mreq_lm;
```

```
  struct sockaddr_in sin_m, sin_lm;
  int sock_m=-1, sock_lm;
  int numread;
  rtp_packet *rp;
  char buffer[MAX_PAYLOAD_SIZE + FIXED_PKTLENGTH];
  char payload_buf[MAX_PAYLOAD_SIZE];
  FILE *fd;
  uint16_t seq;
  uint32_t ssrc, ts;
  int lastPlaySeq=0, numAdSeq=0;

  uint32_t state_index = (uint32_t) index;
  uint32_t m_addr = state_array[state_index].m_addr;
  uint32_t lm_addr = state_array[state_index].lm_addr;
  int addr_len = sizeof (struct sockaddr_in);

  printf("in processRequest(index=%d) m_addr = %x, lm_addr = %x, multicast_port=%d\n", state_
index, m_addr, lm_addr, multicast_port);

  if (state_array[state_index].request_type != LOCAL_PLAY) {
    mreq_m.imr_multiaddr.s_addr = htonl(m_addr);
    mreq_m.imr_interface.s_addr = htonl(INADDR_ANY);

    if((sock_m = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("socket");
      exit(1);
    }

    if(setsockopt(sock_m, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&mreq_m,
                  sizeof(struct ip_mreq)) < 0) {
       perror("setsockopt sock_m");
       exit(1);
    }

    sin_m.sin_addr.s_addr = htonl(m_addr);
    sin_m.sin_port = htons(multicast_port);
    sin_m.sin_family = AF_INET;


    if(bind(sock_m, (struct sockaddr *)&sin_m, sizeof(sin_m)) < 0) {
      perror("bind");
      exit(1);
    }
    if(fcntl(sock_m, F_SETFL, O_NDELAY) < 0) {
      perror("fcntl");
      exit(1);
    }

  }

  mreq_lm.imr_multiaddr.s_addr = htonl(lm_addr);
  mreq_lm.imr_interface.s_addr = htonl(INADDR_ANY);

  if((sock_lm = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("socket");
    exit(1);
  }

  if(setsockopt(sock_lm, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&mreq_lm,
                sizeof(struct ip_mreq)) < 0) {
    perror("setsockopt sock_lm");
    exit(1);
  }
  sin_lm.sin_addr.s_addr = htonl(lm_addr);
```

```
    sin_lm.sin_port = htons(multicast_port);
    sin_lm.sin_family = AF_INET;


    if(bind(sock_lm, (struct sockaddr *)&sin_lm, sizeof(sin_lm)) < 0) {
        perror("bind");
        exit(1);
    }

    /* we want the call to recv to be non-blocking */
    if(fcntl(sock_lm, F_SETFL, O_NDELAY) < 0) {
      perror("fcntl");
      exit(1);
    }

    printf("processRequest(index=%d): request_type=%d\n",
           state_index, state_array[state_index].request_type);

    while (TRUE) {
      switch (state_array[state_index].request_type) {
      case PLAY:

        /* check whether there is information on the socket */
          numread = recv(sock_m, buffer, sizeof(buffer), 0);
          if(numread>0) {
              rp = (rtp_packet *) buffer;
              if (abs(ntohl(rp->RTP_header.seq) - lastPlaySeq) >= numAdSeq-1) {
                 lastPlaySeq = ntohl(rp->RTP_header.seq);
                 numAdSeq = 0;

                 if(sendto(sock_lm, buffer, numread, 0, (struct sockaddr *)&sin_lm, sizeof(str
+ ct sockaddr_in)) < 0) {
                     perror("sendto");
                     exit(1);
                 }
              } else {
                  printf("Ommiting an old packet\n");
              }
          }
          break;

      case COMMERCIAL:
      case LOCAL_PLAY:
        /* default action after we finish the commercial */;
        /* state_array[state_index].request_type =  PLAY; */

        ssrc = random32(1);
        seq = 0;
        ts = 0;
        fd = state_array[state_index].fd;

        while (TRUE) {
          if (state_array[state_index].request_type != COMMERCIAL &&
                state_array[state_index].request_type != LOCAL_PLAY)
              break;

          numread = fread (payload_buf, 1, MAX_PAYLOAD_SIZE, fd);
          if (numread <= 0) {
             fseek(fd, 0, SEEK_SET);
             continue;
          }
          rp = (rtp_packet *) buffer;
          rp->RTP_header.version = RTP_VERSION;
          rp->RTP_header.p = 0;
```

```
        rp->RTP_header.x = 0;
        rp->RTP_header.cc = 0;
        rp->RTP_header.pt = 0;
        rp->RTP_header.seq = htonl(seq);
        rp->RTP_header.ts = htonl(ts);
        rp->RTP_header.ssrc = htonl(ssrc);
        rp->payload_len = htonl(numread);
        memcpy(rp->payload, payload_buf, numread);

        if(sendto(sock_lm, buffer, FIXED_PKTLENGTH + numread, 0, (struct sockaddr *)&sin_lm,
sizeof(struct sockaddr_in)) < 0) {
          perror("sendto");
          exit(1);
        }

        if (ts >= UINT_MAX - numread/8 || seq >= USHRT_MAX - 1) {
          seq = 0;
          ts = 0;
        }
        else {
            seq++;
            ts += numread/8;
        }
        numAdSeq++;

        usleep ((numread/8 - 10) * 1000);
        /* yield to another thread */
        sched_yield();
      }
      break;

    default: /* either STOP or unknown request status */
      if (sock_m >= 0)
        close(sock_m);
      close(sock_lm);
      state_array[state_index].valid = FALSE;
      printf("processRequest(index=%d): request_type=%d.  Returning NULL.\n",
            state_index, state_array[state_index].request_type);

      return NULL;
    } /* switch */

    sched_yield();
  } /* while(true) */
}


int play(int state_index, uint32_t m_addr, uint32_t lm_addr) {
  int status;

 printf("play(state_iindex=%d ...): total_treads=%d, current_state_index=%d\n",
        state_index, total_threads, current_state_index);


  /* If we already have an identical M-to-LM binding */
  if (state_index != -1 && state_array[state_index].m_addr == m_addr) {
        if (state_array[state_index].request_type == PLAY) {
           printf("play(state_index=%d ...): M-to-LM binding already exists\n",
                  state_index);
           return (DUPLICATE);
        }

        if (state_array[state_index].request_type == COMMERCIAL) {
           printf("play(state_index=%d ...): switching from commercial to play\n",
```

```
                state_index);
            state_array[state_index].request_type = PLAY;
            return OK;
        }
    }

    /* If LM is already being used and M is different */
    if (state_index != -1) {

      printf("play(state_index=%d ...); existing LM, new M\n",
            state_index);

      /* NOTE: need synchronization here */

      state_array[state_index].request_type = STOP;

      printf("play(): waiting for thread to terminate\n");
      while(state_array[state_index].valid) { /* wait for thread to terminate */
        /* NOTE: can use condition variables here, or pthread_join() */
        usleep(1000);
      }
      printf("play(): thread has terminated.  Proceeding...\n");

      /* modify the old thread's state entry */
      state_array[state_index].m_addr = m_addr;
      state_array[state_index].request_type = PLAY;
      state_array[state_index].valid = TRUE;

      /* create a new playing thread (using old state entry) */
      if ((status =
           pthread_create(&(state_array[state_index].tid),
                          NULL, processRequest, (void *) state_index)) != 0) {
        printf("play1: can't create thread: error # %d\n", status);
        exit(1);
      }

      return OK;
    }

    /* here, we have a new LM */

    printf("play(state_index=%d): new LM\n", state_index);

    /* NOTE: also synchronization needed below */
    state_index = findNewIndex();

    if (state_index == -1) return STATES_FULL;

    /* create new state entry */
    state_array[state_index].m_addr = m_addr;
    state_array[state_index].lm_addr = lm_addr;
    state_array[state_index].request_type = PLAY;
    state_array[state_index].fd = NULL;
    state_array[state_index].valid = TRUE;

    /* NOTE: synchronize here */
    total_threads++;

    /* create a new playing thread (using old state entry) */
    if ((status =
         pthread_create(&(state_array[state_index].tid),
                        NULL, processRequest, (void *) state_index)) != 0) {
      printf("play2: can't create thread: error # %d\n", status);
      exit(1);
```

```
  }

  return OK;



oid stop(int state_index) {
  printf("stop(state_iindex=%d): total_treads=%d, current_state_index=%d\n",
         state_index, total_threads, current_state_index);

  /* NOTE: synchronize */

  state_array[state_index].request_type = STOP;

  printf("stop(state_index=%d): waiting for thread to terminate\n",
         state_index);
  while(state_array[state_index].valid) { /* wait for thread to terminate */
    /* NOTE: can use condition variables here, or pthread_join() */
    usleep(1000);
  }
  printf("stop(state_index=%d): thread has terminated.  Continuing...\n",
         state_index);

  total_threads--;

  if (state_index == current_state_index) {
    while(state_index >= 0 && !state_array[state_index--].valid) {
      current_state_index--;
    }
  }
}
```

Marconi.java Thu Jun 17 14:13:41 1999 1

```
import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;

public class Marconi {

    private InputStream is;
    private OutputStream os;
    private Socket sock;
    private Hashtable h;

    public Marconi() {
        h = new Hashtable();
    }

    public static void main(String args[]) throws Exception {
        if (args.length != 3) {
            System.out.println("Marconi Server demo program: usage: " +
                               "Marconi <RAS hostname> <RAS port> <RTP port>");
            System.out.println("\nRun this program in place of Marconi " +
        "Server, to manually send requests to the Radio Antenna Server.");
            System.exit(1);
        }

        Marconi m = new Marconi();

        // create a TCP socket for communication with the RAS
        int rtcpPort = Integer.parseInt(args[2]) + 1;
        m.sock = new Socket(args[0], Integer.parseInt(args[1]));
        m.is = m.sock.getInputStream();
        m.os = m.sock.getOutputStream();

        char choice;
        String command;
        String[] commandArray;
        byte[] requestPkt;

        while(true) {
            PromptUser.display();

            choice = PromptUser.getInput();

            switch(choice) {
            case '1': // play
                // get M and LM
                commandArray = PromptUser.getPlayCommand();
                System.out.print("Sending command: ");
                System.out.println("PLAY " + commandArray[0] + " " +
                                   commandArray[1]);
                requestPkt = m.encodeRequest((byte) 1, commandArray[0],
                                             commandArray[1], null);

                m.sendRequest(requestPkt);

                InsertAd iad = new InsertAd(commandArray, rtcpPort, m);
                m.h.put(commandArray[1], iad);
                iad.start();

                break;

            case '2': // stop
                // get LM
                command = PromptUser.getStopCommand();
```

```
                System.out.print("Sending command: ");
                System.out.println("STOP " + command);
                requestPkt = m.encodeRequest((byte) 2, null,
                                             command, null);

                m.sendRequest(requestPkt);

                InsertAd th = (InsertAd)m.h.get(command);
                if (th != null) {
                   th.stop();
                   m.h.remove(command);
                }

                break;

            case '3': //local play
                commandArray = PromptUser.getLocalPlayCommand();
                System.out.print("Sending command: ");
                System.out.println("LOCAL PLAY " + commandArray[0] + " " +
                                   commandArray[1]);

                Schedule sch = new Schedule(commandArray, m);
                sch.start();
                break;

            case 'e':
                System.out.println("\n Try again.\n");
                break;

            case 'q':
            case 'Q':
                m.sock.close();
                System.exit(0);

            default:
                System.out.println("Invalid option.  Try again.");
            } // switch

        } // while true

    } // main


    public synchronized void sendRequest(byte[] requestPkt)
    {
        byte[] statusPkt = new byte[1];
        int bytesRead;

        try {
           // send request to the RAS
           os.write(requestPkt, 0, requestPkt.length);

           // read the status returned by the RAS
           if ((bytesRead = is.read(statusPkt)) < 1) {
               System.err.println("\nError: status not received.\n");
           }
           else {
               System.out.println("RAS returned status: "
                                  + statusMeaning(statusPkt[0]));
           }
         } catch (Exception e) {
           System.out.println(e.getMessage());
         }
    }
```

```
    public byte[] encodeRequest(int requestType, String m_addr,
                                String lm_addr, String path)
    {
        int m = 0, lm = 0, pathLength = (requestType == 3 || requestType
== 4) ? path.length() : 0;

        if (m_addr != null) m = getIntegerIP(m_addr);
        if (lm_addr != null) lm = getIntegerIP(lm_addr);

        // NOTE: we allocate 1 byte for path even if there's no path
        // (this conforms to the RequestPacket structure of the RAS)
        byte[] buffer = new byte[13 + ((requestType == 3 || requestType
== 4) ? pathLength : 1)];


        buffer[0] = (byte) ((requestType << 16) >>> 24);     // request_type
        buffer[1] = (byte) ((requestType << 24) >>> 24);

        buffer[2] = (byte) ((pathLength << 16) >>> 24);     // path_length
        buffer[3] = (byte) ((pathLength << 24) >>> 24);

        // m
        if (requestType == 1) { // play
            buffer[4] = (byte) (m >>> 24);        // 4 bytes of m
            buffer[5] = (byte) ((m << 8) >>> 24);
            buffer[6] = (byte) ((m << 16) >>> 24);
            buffer[7] = (byte) ((m << 24) >>> 24);
        }
        else {
            buffer[4] = 0;
            buffer[5] = 0;
            buffer[6] = 0;
            buffer[7] = 0;
        }

        // lm
        buffer[8] = (byte) (lm >>> 24);        // 4 bytes of lm
        buffer[9] = (byte) ((lm << 8) >>> 24);
        buffer[10] = (byte) ((lm << 16) >>> 24);
        buffer[11] = (byte) ((lm << 24) >>> 24);

        if (requestType == 3 || requestType == 4) { // commercial
            byte[] bytePath = path.getBytes();

            for (int i = 0; i < bytePath.length; i++) {
                buffer[12 + i] = bytePath[i];
            }
        }
        else { // send a zero byte as the path if the path is empty
            buffer[12] = 0;
        }

        return buffer;
    }


    private static int getIntegerIP(String addr) {
        StringTokenizer st = new StringTokenizer(addr, ".");
        int ip = 0;

        try {
            ip =
```

Marconi.java Th[illegible]a 17 14:13:41 1999 4

```
                Integer.parseInt(st.nextToken()) * 256 * 256 * 256 +
                Integer.parseInt(st.nextToken()) * 256 * 256 +
                Integer.parseInt(st.nextToken()) * 256 +
                Integer.parseInt(st.nextToken());
        }
        catch (Exception e) {
            System.out.println("getIntegerIP(): error: " + e.getMessage());
        }

        return ip;
    } // getIntegerIP

    private static String statusMeaning(byte status) {
        switch(status) {
        case 0: return "OK";
        case 1: return "ERROR";
        case 2: return "DUPLICATE";
        case 3: return "STATES_FULL";
        default: return "(UNKNOWN_STATUS=" + status + ")";
        }
    }
}
```

InsertAd.java Thu Jun 17 14:13:54 1999 1

```
import java.net.*;
import java.util.*;
import java.io.*;

public class InsertAd extends Thread
{
    private String mAddress;
    private String mlAddress;
    private int mPort;
    private Marconi marconi;
    private InetAddress mGroup;
    private MulticastSocket ms;
    private Vector adList;
    private int lastSeq;

    public static final int pLength = 24;

    public InsertAd(String [] command, int p, Marconi m) {
        mAddress = command[0];
        mlAddress = command[1];
        mPort = p;
        marconi = m;
        lastSeq = -1;
        adList = new Vector();
    }

    private int decodeInt(byte [] buffer, int start) {
        int result = 0;

        result |= ((buffer[start+3] << 24) >>> 24);
        result |= ((buffer[start+2] << 24) >>> 16);
        result |= ((buffer[start+1] << 24) >>> 8);
        result |= (buffer[start] << 24);

        return result;
    }

    public void run() {
      try {
        mGroup = InetAddress.getByName(mAddress);
        ms = new MulticastSocket(mPort);
        ms.joinGroup(mGroup);

        BufferedReader br =
                new BufferedReader(new FileReader(mAddress + ".lst"));

        String str;
        while ((str = br.readLine()) != null) {
           if (str.equals("")) continue;
           adList.addElement(str);
        }

        br.close();

        byte [] buffer = new byte[pLength];
        int currentAd = 0;
        byte[] requestPkt;

        while (true) {
           DatagramPacket recv = new DatagramPacket(buffer, pLength);
           ms.receive(recv);

           int version = ((buffer[0] << 24) >>> 30);
           int subtype = buffer[0] & 31;
```

```
            int pt = ((buffer[1] << 24) >>> 24);

            short length = 0;
            length |= ((buffer[3] << 24) >>> 24);
            length |= (buffer[2] << 8);

            int ssrc = decodeInt(buffer, 4);
            int cSeq = decodeInt(buffer, 12);
            int remaining = decodeInt(buffer, 16);
            int cLength = decodeInt(buffer, 20);

            /* System.out.println("Received packet:" +
                               " version=" + version +
                               " subtype=" + subtype +
                               " pt=" + pt +
                               " length=" + length +
                               " ssrc=" + ssrc +
                               " seq=" + cSeq +
                               " remaining=" + remaining +
                               " cLength=" + cLength); */

            if (version != 2 || subtype != 1 || length != pLength/4 - 1 ||
                pt != 204 || !(buffer[8] == 'M' && buffer[9] == 'a'
                 && buffer[10] == 'r' && buffer[11] == 'c'))
            {
                System.out.println("Received an invalid packet!");
                continue;
            }

            if (cSeq == lastSeq) {
                //System.out.println("Received a retransmit packet!");
                continue;
            }

            System.out.println("Commercial starting in " +
                remaining + " seconds, lasting " + cLength + " seconds");

            lastSeq = cSeq;
            Thread.sleep(remaining * 1000);

            requestPkt = marconi.encodeRequest((byte) 3, null,
                               mlAddress, (String)adList.elementAt(currentAd));
            marconi.sendRequest(requestPkt);

            System.out.println("Commercial " +
                (String)adList.elementAt(currentAd) + " Started ...");
            Thread.sleep(cLength * 1000);

            requestPkt = marconi.encodeRequest((byte) 1, mAddress,
                                               mlAddress, null);
            marconi.sendRequest(requestPkt);

            System.out.println("... Commercial " +
                (String)adList.elementAt(currentAd) + " Finished");
            currentAd = (currentAd + 1) % adList.size();
        }
      } catch (Exception e) {
            e.printStackTrace();
      }
    }

    public void leave() {
       try { ms.leaveGroup(mGroup); }
       catch (Exception e) { e.printStackTrace(); }
```

```
    }
}
```

IRC.java Thu Jun 17 14:10 1999 1

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.Math.*;

public class IRC {
    /*
     * constants defining buffer sizes, etc.
     */
    public static final int payloadKBytes = 4;
    public static final int payloadBytes = payloadKBytes * 1024;
    public static final int reset = 50;
    public static final int maxBufferEntries = 500;


    private BufferEntry be;
    private CircBuffer cb;

    private byte[] buffer;
    private boolean newSequence = true;
    private int startSeq;
    private int startTs;
    private long startTime;
    private long current_offset;
    private int currentSeq;
    private int ssrc;

    public IRC (){
        cb = new CircBuffer(maxBufferEntries);
    }


    public boolean checkNewSequence(BufferEntry be) {
      if (newSequence || Math.abs(be.seq - currentSeq) >= reset
          || be.ssrc != ssrc) {
          System.out.println("NEW SEQUENCE!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          return true;
      }
      else return false;
    }

    public void initVars() {
      startSeq = be.seq;
      startTs = be.ts;
      ssrc = be.ssrc;
      Date d = new Date();
      startTime = d.getTime();
      current_offset = startTime - startTs + 512 * 20;
      if (newSequence) {
          currentSeq = startSeq;
      }
      newSequence = false;
    }

    private void decodePacket() {
        int b0, b1, b2, b3;

        be = new BufferEntry();

        be.v = (buffer[0] << 24) >>> 30;

        b0 = (buffer[2] << 24) >>> 24;
        b1 = (buffer[3] << 24) >>> 24;
```

```
        be.seq = (b0 << 8) | b1;

        b0 = buffer[4];
        b1 = buffer[5];
        b2 = buffer[6];
        b3 = buffer[7];
        b0 = (b0 << 24) >>> 24;
        b1 = (b1 << 24) >>> 24;
        b2 = (b2 << 24) >>> 24;
        b3 = (b3 << 24) >>> 24;
        be.ts = (b0 << 24) | (b1 << 16) | (b2 << 8) | b3;

        b0 = buffer[8];
        b1 = buffer[9];
        b2 = buffer[10];
        b3 = buffer[11];
        b0 = (b0 << 24) >>> 24;
        b1 = (b1 << 24) >>> 24;
        b2 = (b2 << 24) >>> 24;
        b3 = (b3 << 24) >>> 24;
        be.ssrc = (b0 << 24) | (b1 << 16) | (b2 << 8) | b3;

        b0 = buffer[16];
        b1 = buffer[17];
        b2 = buffer[18];
        b3 = buffer[19];
        b0 = (b0 << 24) >>> 24;
        b1 = (b1 << 24) >>> 24;
        b2 = (b2 << 24) >>> 24;
        b3 = (b3 << 24) >>> 24;
        be.length = (b0 << 24) | (b1 << 16) | (b2 << 8) | b3;

        be.payload = new byte[be.length];
        for (int i = 20; i < be.length + 20; i++) {
            be.payload[i - 20] = buffer[i];
        }
    }

    public static void main(String[] args) {
        try {

            IRC irc = new IRC();

            if (args.length != 2) {
                System.out.println("Usage : IRC <multicast addr> <port>");
                return;
            }

            String mcastAddr = args[0];
            int port = Integer.parseInt(args[1]);

            // join a Multicast group
            InetAddress group = InetAddress.getByName(mcastAddr);
            MulticastSocket s = new MulticastSocket(port);
            s.joinGroup(group);

            irc.buffer = new byte[payloadBytes + 20];

            irc.cb.start();

            while(true) {
                DatagramPacket recv = new DatagramPacket(irc.buffer, irc.buffer.length);
                s.receive(recv);
                irc.decodePacket();
```

```
                System.out.println(" received packet:" +
                                   " seq=" + irc.be.seq +
                                   " ts=" + irc.be.ts +
                                   " length=" + irc.be.length +
                                   " ssrc=" + irc.be.ssrc);

                if (irc.checkNewSequence(irc.be))
                    irc.initVars();

                irc.currentSeq = irc.be.seq;
                irc.be.offset = irc.current_offset;

                Date date = new Date();
                long time = date.getTime();

                irc.cb.enqueue(irc.be);


            } // while true

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    } // main
} // IRC
```

Marconi.java Thu Jun 17 14:14:38 1999 1

```
import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.*;

public class Marconi {

    private InputStream is;
    private OutputStream os;
    private Socket sock;
    private Hashtable h;

    public Marconi() {
        h = new Hashtable();
    }

    public static void main(String args[]) throws Exception {
        if (args.length != 3) {
            System.out.println("Marconi Server demo program: usage: " +
                               "Marconi <RAS hostname> <RAS port> <RTP port>");
            System.out.println("\nRun this program in place of Marconi " +
        "Server, to manually send requests to the Radio Antenna Server.");
            System.exit(1);
        }

        Marconi m = new Marconi();

        // create a TCP socket for communication with the RAS
        int rtcpPort = Integer.parseInt(args[2]) + 1;
        m.sock = new Socket(args[0], Integer.parseInt(args[1]));
        m.is = m.sock.getInputStream();
        m.os = m.sock.getOutputStream();

        char choice;
        String command;
        String[] commandArray;
        byte[] requestPkt;

        while(true) {
            PromptUser.display();

            choice = PromptUser.getInput();

            switch(choice) {
            case '1': // play
                // get M and LM
                commandArray = PromptUser.getPlayCommand();
                System.out.print("Sending command: ");
                System.out.println("PLAY " + commandArray[0] + " " +
                                   commandArray[1]);
                requestPkt = m.encodeRequest((byte) 1, commandArray[0],
                                             commandArray[1], null);

                m.sendRequest(requestPkt);

                InsertAd iad = new InsertAd(commandArray, rtcpPort, m);
                m.h.put(commandArray[1], iad);
                iad.start();

                break;

            case '2': // stop
                // get LM
                command = PromptUser.getStopCommand();
```

```
                System.out.print("Sending command: ");
                System.out.println("STOP " + command);
                requestPkt = m.encodeRequest((byte) 2, null,
                                             command, null);

                m.sendRequest(requestPkt);

                InsertAd th = (InsertAd)m.h.get(command);
                if (th != null) {
                   th.stop();
                   m.h.remove(command);
                }

                break;

            case '3': //local play
                commandArray = PromptUser.getLocalPlayCommand();
                System.out.print("Sending command: ");
                System.out.println("LOCAL PLAY " + commandArray[0] + " " +
                                   commandArray[1]);

                Schedule sch = new Schedule(commandArray, m);
                sch.start();
                break;

            case 'e':
                System.out.println("\n Try again.\n");
                break;

            case 'q':
            case 'Q':
                m.sock.close();
                System.exit(0);

            default:
                System.out.println("Invalid option.  Try again.");
            } // switch

        } // while true

    } // main


    public synchronized void sendRequest(byte[] requestPkt)
    {
        byte[] statusPkt = new byte[1];
        int bytesRead;

        try {
           // send request to the RAS
           os.write(requestPkt, 0, requestPkt.length);

           // read the status returned by the RAS
           if ((bytesRead = is.read(statusPkt)) < 1) {
               System.err.println("\nError: status not received.\n");
           }
           else {
               System.out.println("RAS returned status: "
                                  + statusMeaning(statusPkt[0]));
           }
         } catch (Exception e) {
           System.out.println(e.getMessage());
         }
    }
```

```
    public byte[] encodeRequest(int requestType, String m_addr,
                                String lm_addr, String path)
    {
        int m = 0, lm = 0, pathLength = (requestType == 3 || requestType
== 4) ? path.length() : 0;

        if (m_addr != null) m = getIntegerIP(m_addr);
        if (lm_addr != null) lm = getIntegerIP(lm_addr);

        // NOTE: we allocate 1 byte for path even if there's no path
        // (this conforms to the RequestPacket structure of the RAS)
        byte[] buffer = new byte[13 + ((requestType == 3 || requestType
== 4) ? pathLength : 1)];


        buffer[0] = (byte) ((requestType << 16) >>> 24);     // request_type
        buffer[1] = (byte) ((requestType << 24) >>> 24);

        buffer[2] = (byte) ((pathLength << 16) >>> 24);     // path_length
        buffer[3] = (byte) ((pathLength << 24) >>> 24);

        // m
        if (requestType == 1) { // play
            buffer[4] = (byte) (m >>> 24);        // 4 bytes of m
            buffer[5] = (byte) ((m << 8) >>> 24);
            buffer[6] = (byte) ((m << 16) >>> 24);
            buffer[7] = (byte) ((m << 24) >>> 24);
        }
        else {
            buffer[4] = 0;
            buffer[5] = 0;
            buffer[6] = 0;
            buffer[7] = 0;
        }

        // lm
        buffer[8] = (byte) (lm >>> 24);        // 4 bytes of lm
        buffer[9] = (byte) ((lm << 8) >>> 24);
        buffer[10] = (byte) ((lm << 16) >>> 24);
        buffer[11] = (byte) ((lm << 24) >>> 24);

        if (requestType == 3 || requestType == 4) { // commercial
            byte[] bytePath = path.getBytes();

            for (int i = 0; i < bytePath.length; i++) {
                buffer[12 + i] = bytePath[i];
            }
        }
        else { // send a zero byte as the path if the path is empty
            buffer[12] = 0;
        }

        return buffer;
    }


    private static int getIntegerIP(String addr) {
        StringTokenizer st = new StringTokenizer(addr, ".");
        int ip = 0;

        try {
            ip =
```

```
                Integer.parseInt(st.nextToken()) * 256
                Integer.parseInt(st.nextToken()) * 256 * 256 +
                Integer.parseInt(st.nextToken()) * 256 +
                Integer.parseInt(st.nextToken());

        }
        catch (Exception e) {
            System.out.println("getIntegerIP(): error: " + e.getMessage());
        }

        return ip;
    } // getIntegerIP

    private static String statusMeaning(byte status) {
        switch(status) {
        case 0: return "OK";
        case 1: return "ERROR";
        case 2: return "DUPLICATE";
        case 3: return "STATES_FULL";
        default: return "(UNKNOWN_STATUS=" + status + ")";
        }
    }
}
```

RsSendRTCP.java Thu [redacted] 17 .4:15:14 1999 1

```
import java.net.*;
import java.util.*;
import java.io.*;

public class RsSendRTCP extends Thread
{
    private String mAddress;
    private int mPort;
    private String scheduleFile;
    private int ssrc;
    private InetAddress mGroup;
    private MulticastSocket ms;
    private int cSeq;
    private Hashtable hTable;

    public static final byte version = 2;
    public static final int plength = 24;
    public static final int retransmit = 3;
    public static final byte subtype = 1;
    public static final int maxHash = 7 * 86400;

    public RsSendRTCP(String addr, int p, String file)
    {
        mAddress = addr;
        mPort = p;
        scheduleFile = file;
        ssrc = (int) (java.lang.Math.random() * 10000);
        cSeq = 0;
        hTable = new Hashtable();
    }

    public void run() {
      try {
        mGroup = InetAddress.getByName(mAddress);
        ms = new MulticastSocket(mPort);
        ms.joinGroup(mGroup);

        BufferedReader br = new BufferedReader(new FileReader(scheduleFile));

        String str;
        int lineNum = 0;
        while ((str = br.readLine()) != null) {
            lineNum++;

            if (str.equals("")) continue;

            StringTokenizer st = new StringTokenizer(str, " \t");
            if (st.countTokens() != 5) {
                System.err.println("Illegal number of parameters on line "
                                   + lineNum + " of " + scheduleFile);
                continue;
            }

            int day = Integer.parseInt(st.nextToken());
            int hour = Integer.parseInt(st.nextToken());
            int minute = Integer.parseInt(st.nextToken());
            int second = Integer.parseInt(st.nextToken());
            int duration = Integer.parseInt(st.nextToken());
            day--;
            int hash = day * 86400 + hour * 3600 + minute * 60 + second;

            // System.out.println("Inserting " + day + " " + hour + " " +
            //          minute + " " + second + " " + duration + " " + hash);
            hTable.put(new Integer(hash), new Integer(duration));
```

RsSendRTCP.java Jun 17 14:15:14 1999

```
        }
        br.close();

        while (true) {

          Calendar myCalendar = Calendar.getInstance();
          int day = myCalendar.get(Calendar.DAY_OF_WEEK);
          int hour = myCalendar.get(Calendar.HOUR_OF_DAY);
          int minute = myCalendar.get(Calendar.MINUTE);
          int second = myCalendar.get(Calendar.SECOND);
          day--;
          int hash = (day * 86400 + hour * 3600 + minute * 60 +
                      second + retransmit + 1) % maxHash;

          // System.out.println("Now is " + day + " " + hour + " " + minute +
          //     " " + second);

          Integer d;
          if ( (d = (Integer)hTable.get(new Integer(hash))) != null) {
             int duration = d.intValue();

             // create an RTCP packet
             byte [] buffer = new byte[pLength];
             buffer[0] = (byte) (version << 6);    // version, p
             buffer[0] |= subtype;                 // subtype
             buffer[1] = (byte)204;                // pt

             short l = pLength/4 - 1;
             buffer[2] = (byte) (l >>> 8);         // length (higher byte)
             buffer[3] = (byte) ((l << 8) >>> 8); // length (lower byte)

             // ssrc
             buffer[4] = (byte) (ssrc >>> 24);
             buffer[5] = (byte) ((ssrc << 8) >>> 24);
             buffer[6] = (byte) ((ssrc << 16) >>> 24);
             buffer[7] = (byte) ((ssrc << 24) >>> 24);

             // name
             buffer[8] = 'M';
             buffer[9] = 'a';
             buffer[10] = 'r';
             buffer[11] = 'c';

             // cseq
             buffer[12] = (byte) (cSeq >>> 24);
             buffer[13] = (byte) ((cSeq << 8) >>> 24);
             buffer[14] = (byte) ((cSeq << 16) >>> 24);
             buffer[15] = (byte) ((cSeq << 24) >>> 24);
             cSeq = (int) ((cSeq == 32767) ? 0 : cSeq + 1);

             // length of commercial
             buffer[20] = (byte) (duration >>> 24);
             buffer[21] = (byte) ((duration << 8) >>> 24);
             buffer[22] = (byte) ((duration << 16) >>> 24);
             buffer[23] = (byte) ((duration << 24) >>> 24);

             for (int i = retransmit+1; i>0; i--) {

                 // time remaining to commercial
                 buffer[16] = (byte) (i >>> 24);
                 buffer[17] = (byte) ((i << 8) >>> 24);
                 buffer[18] = (byte) ((i << 16) >>> 24);
                 buffer[19] = (byte) ((i << 24) >>> 24);
```

```
                System.out.println(
                        "Sending alert for commercial break starting in " +
                        i + " seconds, lasting " + duration + " seconds");
                DatagramPacket dp = new DatagramPacket
                            (buffer, pLength, mGroup, mPort);
                ms.send(dp);
                Thread.sleep(1000);
            }

          }

          Thread.sleep(1000);
        }

      } catch (Exception e) {
          e.printStackTrace();
      }
    }

    public void leave() {
       try { ms.leaveGroup(mGroup); }
       catch (Exception e) {
          e.printStackTrace();
       }
    }

}
```

Schedule.java Thu Jun 17 1 :15:36 1999 1

```
import java.io.*;
import java.util.*;
import java.lang.*;
import java.text.*;

public class Schedule extends Thread {

private String lmAddress;
private Marconi marconi;
public static String scheduleFile;
public static int[] dateArray = new int[4];

public Schedule (String [] command, Marconi m) {
  lmAddress = command[0];
  scheduleFile = command[1];
  marconi = m;
}

public void run() {
  try {
    byte[] requestPkt;
    boolean flag;

    while (true) {
      BufferecReader br = new BufferedReader(new FileReader(scheduleFile));

      int[] array = new int[8];
      String str;
      while ((str = br.readLine()) != null) {
        int index = 0;
        for (int i=0; i<8; i++) {
          int endIndex = str.indexOf(' ', index);
          array[i] = Integer.parseInt(str.substring(index, endIndex));
          index = endIndex+1;
        }
        String fileName = str.substring(index, str.length()-1);
        getCurrentDate();

        if (isBetween(array)) {
          // start playing
          if (isAddress(fileName)) flag = false;
          else flag = true;

          System.out.println("fileName: "+fileName);
          if (flag)
            requestPkt = marconi.encodeRequest((byte)4, null, lmAddress, fileName);
          else
            requestPkt = marconi.encodeRequest((byte)1, fileName, lmAddress, null);

        marconi.sendRequest(requestPkt);

/*      System.out.println ("now in date range:");
        for (int i=0; i<8; i++) {
          System.out.print(array[i] + " ");
        }
        System.out.println();
        if (flag)
          System.out.println("Playing file "+fileName);
        else
          System.out.println("Redirecting packets from "+fileName); */

        while (true) {
          Thread.sleep(1000);
          getCurrentDate();
```

```
          if (!isBetween(array)) break;
        }
        // stop playing
          requestPkt = marconi.encodeRequest((byte)2, null, lmAddress, null);
          marconi.sendRequest(requestPkt);
        } // if
      }
    } //while
  }
  catch (Exception e) {
    System.out.println(e.getMessage());
  }
}

public boolean isAddress (String str) {
  int count=0;
  int index=0;
  while ((index=str.indexOf('.', index)) != -1) {
    count++;
    index++;
  }
  if (count == 3) return true;
  else return false;
}

public static void main(String args[]) throws Exception {

  if (args.length != 1) {
    System.out.println("Usage : Schedule <schedule data file>");
    return;
  }
  String scheduleFile = args[0];
  while (true) {
    BufferedReader br = new BufferedReader(new FileReader(scheduleFile));

    int[] array = new int[8];
    String str;
    while ((str = br.readLine()) != null) {
      int index = 0;
    for (int i=0; i<8; i++) {
      int endIndex = str.indexOf(' ', index);
      array[i] = Integer.parseInt(str.substring(index, endIndex));
      index = endIndex+1;
    }
    String fileName = str.substring(index, str.length()-1);
    String localName = "LOCAL";
    getCurrentDate();

    if (isBetween(array)) {
      // PLAY
      System.out.println ("current time:");
      for (int i=0; i<4; i++) {
        System.out.print(dateArray[i] + " ");
      }
      System.out.println();
      System.out.println ("date range:");
      for (int i=0; i<8; i++) {
        System.out.print(array[i] + " ");
      }
      System.out.println();
      System.out.println("Playing file "+fileName);
        while (true) {
          Thread.sleep(1000);
          getCurrentDate();
```

```
          if (!isBetween(array)) break;
        }
      // STOP PLAYING
    } // if
    }
  } //while
} // main

public static void getCurrentDate() {
  Date myDate = new Date();
  Calendar myCalendar = Calendar.getInstance();
  dateArray[0] = myCalendar.get(Calendar.DAY_OF_WEEK);
  dateArray[1] = myCalendar.get(Calendar.HOUR_OF_DAY);
  dateArray[2] = myCalendar.get(Calendar.MINUTE);
  dateArray[3] = myCalendar.get(Calendar.SECOND);
}

public static boolean isBetween(int[] array) {

  for (int i=array[0]; ;i=(i+1)%7) {
    if (i == dateArray[0]) {
      if (i != array[0] && i != array[4])
        return true; //between 2 days

      if (i == array[0]) { //sometime on the left boundary day
        if (dateArray[1] < array[1]) // if hour is earlier
          return false;
        if (dateArray[1] == array[1]) { // figure out the minutes
          if (dateArray[2] < array[2]) return false;
          if (dateArray[2] == array[2]) { // figure out the seconds
            if (dateArray[3] < array[3]) return false;
            if (dateArray[3] == array[3] || array[0] != array[4])
              return true;
          }
          else if (dateArray[0] != array[4]) return true;
        }
        else if (dateArray[0] != array[4]) return true;
      }
      if (i == array[4]) { //sometime on the right boundary day
        if (dateArray[1] > array[5]) // if time is later
          return false;
        else if (dateArray[1] < array[5]) //has already been checked for early
          return true;
        else { // hours are equal
          if (dateArray[2] < array[6]) return true;
          if (dateArray[2] > array[6]) return false;
          //minutes are equal
          if (dateArray[3] > array[7]) return false;
          return true;
        }
      }
      break;
    }
    if (i == array[4]) break;
  }
  return false;
}

} // class
```

```
CircBuffer.java          Thu Dec 17 14:15:58 1999        1

import java.lang.*;
import java.util.*;
import java.io.*;

public class CircBuffer extends Thread {
    private int maxSize;
    private int currentIndex;
    private BufferEntry[] array;
    private int delay = 100;

    private AudioOutputStream aos;

    private int seq = 0;

    private void write(byte[] payload) {
        try {
            aos.write(payload, 0, payload.length);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public CircBuffer(int maxBufferSize) {
        maxSize = maxBufferSize;
        currentIndex = 0;
        array = new BufferEntry[maxSize];
    }

    public void enqueue (BufferEntry be) {
        array[be.seq % maxSize] = be;
    }

    public void incrSeq () {
        seq = (seq == 32767) ? 0 : seq + 1;
    }

    public void run() {
        int i;
        aos = new AudioOutputStream();

        while(true) {
          for (i=0; i<maxSize; i++) {
              if (array[seq % maxSize] != null) {
                  break;
              }
              else incrSeq();
          }

          if (i == maxSize) {
              continue;
          }

          seq = array[seq % maxSize].seq;
          Date date = new Date();
          long time = date.getTime();

          if (array[seq % maxSize].ts + array[seq % maxSize].offset + delay < time) {
            System.err.println("late packet: ts = " + array[seq % maxSize].ts + " current tim
e = " + time + " offset = " + array[seq % maxSize].offset);
            array[seq % maxSize] = null;
            continue;
          }
```

CircBuffer.java Thu Jun 17 14:15:58 1999 2

```
            if (array[seq % maxSize].ts + array[seq % maxSize].offset > time) {
                incrSeq();
                //System.out.println("Packet " + seq + " is too early to play");
                continue;
            }


            System.err.println("writing packet: seq=" + seq+ " ssrc=" + array[seq % maxSize]
+ rc + " ts=" + array[seq % maxSize].ts);
            write(array[seq % maxSize].payload);

            array[seq % maxSize] = null;
            Thread.yield();

        } // while true
    } // run
}
```

BufferEntry.java Thu Jun 1' 14:16:10 1999 1

```
class BufferEntry {
    public int v, ssrc, ts, seq, length;
    public long offset;
    public byte[] payload;
}
```

PromptUser.java Thu [redacted] 17 14:16:37 1999 1

```
/*
IRC uses local class promptUser, which takes care of console input/output.
It displays a menu, gets user input and displays the return values of
invoked remote methods.
*/

import java.util.*;
import java.io.*;

public class PromptUser {
    static String command;
    static BufferedReader br;
    static String mAddr;
    static String lmAddr;
    static String path;

    /**
     *  Display the menu
     */
    public static void display() {
        System.out.println("\n\n--------- Menu ----------");
        System.out.println("1.  play");
        System.out.println("2.  stop");
        System.out.println("3.  local play");
        System.out.println("Q.  quit");
        System.out.print("\nSelect an option:  ");
    }


    // read a number corresponding to user's option and convert it to
    // character
    public static char getInput() {
        char choice = 'e';
        try{
            br = new BufferedReader(new InputStreamReader(System.in));
            choice = (char)br.read();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        return choice;
    }


    public static String[] getPlayCommand() {
        System.out.print("Enter <M> <LM>: ");

        try{
            br = new BufferedReader(new InputStreamReader(System.in));
            command = br.readLine();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        StringTokenizer st = new StringTokenizer(command);
        int argc = st.countTokens();
        String [] array = new String[argc];

        for (int i=0; i<argc; i++)
            array[i] = st.nextToken();
```

PromptUser.java [redacted]u Jun 17 14:16:37 1999 2

```
        return array;
    }

    public static String[] getLocalPlayCommand() {
       System.out.print("Enter <LM> <SCHEDULE FILE>: ");

        try{
            br = new BufferedReader(new InputStreamReader(System.in));
            command = br.readLine();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        String[] array = new String[2];
        StringTokenizer st = new StringTokenizer(command);

        for (int i = 0; st.hasMoreTokens() && i < 2; i++) {
            array[i] = st.nextToken();
        }

        return array;
    }


    public static String getStopCommand() {
        System.out.print("Enter <LM>: ");

        try{
            br = new BufferedReader(new InputStreamReader(System.in));
            command = br.readLine();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }

        return command;
    }
}
```

```
/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"
#include <memory.h>

static char rcsid[] = "$Id: md5c.c,v 1.1 1997/12/17 12:52:36 hgs Exp $";

/* Constants for MD5Transform routine.
 */

#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
```

```
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \                                    ,*
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \                        :!
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                        /* context */
unsigned char *input;                                /* input block */
unsigned int inputLen;                     /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3)) < ((UINT4)inputLen << 3))
```

```
    context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Transform as many times as possible.  */
  if (inputLen >= partLen) {
    memcpy
      ((POINTER)&context->buffer[index], (POINTER)input, partLen);
    MD5Transform (context->state, context->buffer);

    for (i = partLen; i + 63 < inputLen; i += 64)
      MD5Transform (context->state, &input[i]);

    index = 0;
  }
  else
    i = 0;

  /* Buffer remaining input */
  memcpy
    ((POINTER)&context->buffer[index], (POINTER)&input[i],
    inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (digest, context)
unsigned char digest[16];                         /* message digest */
MD5_CTX *context;                                       /* context */
{
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Save number of bits */
  Encode (bits, context->count, 8);

  /* Pad out to 56 mod 64. */
  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD5Update (context, PADDING, padLen);

  /* Append length (before padding) */
  MD5Update (context, bits, 8);


  /* Store state in digest */
  Encode (digest, context->state, 16);

  /* Zeroize sensitive information. */
  memset ((POINTER)context, 0, sizeof (*context));
} /* MD5final */


/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);
```

```
/* Round 1 */
FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

/* Round 2 */
GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

/* Round 3 */
HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

/* Round 4 */
II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
```

```
md5c.c          Thu Jun 17 14:51:51 1999        5

  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Zeroize sensitive information. */
  memset ((POINTER)x, 0, sizeof (x));
}

/*
 * Encodes input (UINT4) into output (unsigned char). Assumes len is
 * a multiple of 4.
 */
static void Encode(output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/*
 * Decodes input (unsigned char) into output (UINT4). Assumes len is
 * a multiple of 4.
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}
```

random32.c Thu Jun 17 14:17:17 1999 1

```
/*
* Generate a random 32-bit quantity.
*/
#include <sys/time.h>           /* gettimeofday() */
#include <unistd.h>             /* get..() */
#include <string.h>             /* strncpy() */
#include <stdlib.h>             /* atoi() */
#include <stdio.h>              /* printf() */
#include <sys/utsname.h>        /* uname() */
#include <time.h>               /* clock() */
#include "global.h"             /* from RFC 1321 */
#include "md5.h"                /* from RFC 1321 */

extern long gethostid(void);

static char rcsid[] = "$Id: random32.c,v 1.2 1998/09/29 16:31:37 hgs Exp $";

#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
typedef unsigned long  u_int32;

static u_int32 md_32(char *string, int length)
{
  MD_CTX context;
  union {
    char   c[16];
    u_int32 x[4];
  } digest;
  u_int32 r;
  int i;

  MDInit (&context);
  MDUpdate (&context, (unsigned char *)string, length);
  MDFinal ((unsigned char *)&digest, &context);
  /* XOR the four parts into one word */
  for (i = 0, r = 0; i < 3; i++) r ^= digest.x[i];
  return r;
} /* md_32 */



/*
* Return random unsigned 32-bit quantity.
*/
uint32_t random32(int type)
{
  struct {
    int     type;
    struct  timeval tv;
    clock_t cpu;
    pid_t   pid;
    u_long  hid;
    uid_t   uid;
    gid_t   gid;
    struct  utsname name;
  } s;

  gettimeofday(&s.tv, 0);
  s.type = type;
  s.cpu  = clock();
  s.pid  = getpid();
  s.hid  = gethostid();
```

```
  s.uid  = getuid();
  s.gid  = getgid();
  uname(&s.name);

  return md_32((char *)&s, sizeof(s));
} /* random32 */
```

AudioOutputStream.java Thu Jun 17 14:17:38 1999 1

```
import java.io.*;
import java.util.Date;

/**
 * An audio output stream is an output stream for writing data to a
 * (possible dummy) audio device.
 * <P>
 * The stream records performance statistics, such as the milliseconds of
 * "dead air", * the number of writes that were late, etc.  Statistics are
 * kept following the first write() invocation.
 * <P>
 * For accuracy, the class keeps statistics internally in microseconds,
 * but exports them in milliseconds.
 * <P>
 * On architectures supporting the "/dev/audio" device file (Solaris, Linux),
 * the AudioOutputStream will also write the data to the audio device.
 *
 * @author Alexander V. Konstantinou
 * @version $Revision: 1.5 $
 */
public class AudioOutputStream extends OutputStream {

  /**
   * microseconds per byte for 8 KHz, 8-bit MLAW samples
   */
  public static final long mspb_8KHz_8bit_MLAW = 125;

  private long expiresMicros;
  private long microsPerByte;
  private long missedMicros;
  private long lateWrites;
  private FileOutputStream fostream = null;

  /**
   * Create new AudioOutputStream with default encoding 8 KHz, 8-bit MLAW
   * <P>
   * On architectures supporting a "/dev/audio" device file, the constructor
   * will attempt to open the device for writing (but will not fail if
   * this is refused).
   */
  public AudioOutputStream() {
    super();
    microsPerByte = mspb_8KHz_8bit_MLAW;

    // Try to open the /dev/audio device file
    try {
      fostream = new FileOutputStream("/dev/audio");
    } catch (Exception e) {
      System.err.println("!!!!!!! Can't open /dev/audio !!!!!!!!!");
      fostream = null;
    }
  }

  /**
   * Create new AudioOutputStream for an unknown encoding which
   * expands a byte of data to "microsecondsPerByte" <B>micro</B>seconds
   * <P>
   * Note that the value is in <B>microseconds<B>, not milliseconds.
   * <P>
   * No attempt will be made to open the "/dev/audio" device file due
   * to the unknown format.
   */
  public AudioOutputStream(long microsecondsPerByte) {
    super();
```

AudioOutputStream.jav Thu Jun 17 14:17:38 1999

```
    microsPerByte = microsecondsPerByte;
    fostream = null; // unknown encoding is not written to audio device
  }

  /**
   * Resets audio play statistics
   */
  public void resetStats() {
    expiresMicros = 0;
    missedMicros = 0;
    lateWrites = 0;
  }

  /**
   * Updates statistics for "count" bytes were buffered for audio output
   */
  private void logWriteBytes(int count) {
    long currentMicros = System.currentTimeMillis() * 1000;
    if (currentMicros > expiresMicros) {
      if (expiresMicros > 0) {
        missedMicros += currentMicros - expiresMicros;
        ++lateWrites;
      }
      expiresMicros = currentMicros + count * microsPerByte;
    } else {
      expiresMicros += count * microsPerByte;
    }
  }

  /**
   * Writes the specified byte to the audio device.
   *
   * Use of this method is discouraged, as the overhead for obtaining
   * the current time will be significant compared to the play time
   * for one byte.
   */
  public void write(int b) throws IOException {
    logWriteBytes(1);
    if (fostream != null) fostream.write(b);
  }

  /**
   * Writes b.length bytes from the specified byte array to the audio
   * device.
   *
   * Avoid very small byte arrays for the same reason as write(int b).
   */
  public void write(byte b[]) throws IOException {
    logWriteBytes(b.length);
    if (fostream != null) fostream.write(b);
  }

  /**
   * Writes len bytes from the specified byte array starting at offset off
   * to the audio device.
   *
   * Avoid very small lengths (len) for the same reason as write(int b).
   */
  public void write(byte b[], int off, int len) throws IOException {
    logWriteBytes(len);
    if (fostream != null) fostream.write(b);
  }

  /**
```

AudioOutputStream.java Thu Jun 17 14:17:38 1999 3

```
    * Flushes this output stream and forces any buffered output bytes to
    * be written out.  Currently a no-op (does nothing).
    */
   public void flush() throws IOException {
   }

   /**
    * Closes this output stream and releases any system resources associated
    * with this stream.
    */
   public void close() throws IOException {
     resetStats();
     if (fostream != null) {
       fostream.close();
       fostream = null;
     }
   }

   /**
    * Returns the number of milliseconds of "dead air" up to the last
    * write.
    *
    * Succesive invocations of this method without other intervening
    * AudioOutputStream method invocations will return the same value.
    */
   public long getMissedMillisAtLastWrite() {
     return(missedMicros / 1000);
   }

   /**
    * Returns the number of milliseconds of "dead air" accumulated so far.
    *
    * Succesive invocations of this method may not return the same value if
    * there is no buffered data for output, as this method returns the
    * total so far (current time).
    */
   public long getMissedMillis() {
     long currentMicros = System.currentTimeMillis() * 1000;

     if (currentMicros > expiresMicros) {
       if (expiresMicros == 0)  // special case - no writes so far
         return 0;
       else
         return( (missedMicros + (currentMicros - expiresMicros)) / 1000);
     } else {
       return(missedMicros/1000);
     }
   }

   /**
    * Returns the number of late writes counted so far
    */
   public long getLateWriteCount() {
     return(lateWrites);
   }

   /**
    * Returns the expiration time in milliseconds of the current audio
    * playing.
    *
    * If no audio is currently buffered, will return the time
    * when the last byte completed playing.
    * Time is represented as microseconds between the current time and
    * midnight, January 1, 1970 UTC.  This measure is system dependent,
```

AudioOutputStream.java Thu Jun 17 14:17:38 1999 3

```
   * Flushes this output stream and forces any buffered output bytes to
   * be written out.  Currently a no-op (does nothing).
   */
  public void flush() throws IOException {
  }

  /**
   * Closes this output stream and releases any system resources associated
   * with this stream.
   */
  public void close() throws IOException {
    resetStats();
    if (fostream != null) {
      fostream.close();
      fostream = null;
    }
  }

  /**
   * Returns the number of milliseconds of "dead air" up to the last
   * write.
   *
   * Succesive invocations of this method without other intervening
   * AudioOutputStream method invocations will return the same value.
   */
  public long getMissedMillisAtLastWrite() {
    return(missedMicros / 1000);
  }

  /**
   * Returns the number of milliseconds of "dead air" accumulated so far.
   *
   * Succesive invocations of this method may not return the same value if
   * there is no buffered data for output, as this method returns the
   * total so far (current time).
   */
  public long getMissedMillis() {
    long currentMicros = System.currentTimeMillis() * 1000;

    if (currentMicros > expiresMicros) {
      if (expiresMicros == 0)  // special case - no writes so far
        return 0;
      else
        return( (missedMicros + (currentMicros - expiresMicros)) / 1000);
    } else {
      return(missedMicros/1000);
    }
  }

  /**
   * Returns the number of late writes counted so far
   */
  public long getLateWriteCount() {
    return(lateWrites);
  }

  /**
   * Returns the expiration time in milliseconds of the current audio
   * playing.
   *
   * If no audio is currently buffered, will return the time
   * when the last byte completed playing.
   * Time is represented as microseconds between the current time and
   * midnight, January 1, 1970 UTC.  This measure is system dependent,
```

AudioOutputStream.java Thu Jun 17 14:17:38 1999

```
        long sleepMillis = (long) (2 * (aos.getBufferExpirat
lis()) * Math.random());
        if (sleepMillis > 0) {
          System.out.println("Sleeping for " + sleepMillis + " ...");
          Thread.sleep(sleepMillis);
        }
      } while (n != -1);

      aos.printStats();
      aos.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

} // class AudioOutputStream
```

TextEdit.java Thu Jun 17 14:18:26 1999 1

```
import java.awt.*;
import java.io.*;

public class TextEdit extends Frame {
  public TextEdit(String [] commandArray, Marconi m) {
    this.m = m;
    this.commandArray = commandArray;
    setTitle("TextEdit");
    Panel p1 = new Panel();
    p1.setLayout(new FlowLayout());
    p1.add(new Label("filename: "));
    filename = new TextField(commandArray[1], 20);
    p1.add(filename);
    add("Center", p1);
    Panel p2 = new Panel();
    p2.setLayout(new FlowLayout());
    p2.add(createButton = new Button ("Create"));
    p2.add(finishButton = new Button ("Finish"));
    add("South", p2);
  }


  public boolean handleEvent(Event event) {
    if (event.id == Event.ACTION_EVENT && event.target == createButton) {
      String fname = filename.getText();
      if (fname.equals("")) return true;
/*      boolean flag = false;
      try {
        BufferedReader br = new BufferedReader(new FileReader(fname));
      }
      catch (FileNotFoundException e) {
        flag = true;
      }
      if (!flag) {
        System.out.println ("file "+fname+" already exists");
        return true;
      } */
      Frame ne = new NewEdit(fname, this);
      ne.resize(200, 150);
      this.hide();
      ne.show();
      return true;
    }
    if (event.id == Event.ACTION_EVENT && event.target == finishButton) {
      dispose();
      m.sendSchedule(commandArray, Thread.currentThread());
      Thread.currentThread().suspend();
      return true;
    }
    return super.handleEvent(event);
  }

/*
  public static void main(String[] args) {
    String [] str = new String[];
    Frame f = new TextEdit(str);
    f.setSize(300, 150);
    f.show();
  }
*/

private TextField filename;
private Button createButton, finishButton;
private String[] commandArray;
```

```
private Marconi m;
}
```

Schedule.java Thu Jun 17 14 18:49 1999 1

```
import java.io.*;
import java.util.*;
import java.lang.*;
import java.text.*;

public class Schedule extends Thread {

private String lmAddress;
private Marconi marconi;
public static String scheduleFile;
public static int[] dateArray = new int[4];

public Schedule (String [] command, Marconi m) {
  lmAddress = command[0];
  scheduleFile = command[1];
  marconi = m;
}

public void run() {
  try {
    byte[] requestPkt;
    boolean flag;

    while (true) {
      BufferedReader br = new BufferedReader(new FileReader(scheduleFile));

      int[] array = new int[8];
      String str;
      while ((str = br.readLine()) != null) {
        int index = 0;
        StringTokenizer st = new StringTokenizer(str, " ");
        for (int i=0; i<8; i++) {
         /* int endIndex = str.indexOf(' ', index);
          array[i] = Integer.parseInt(str.substring(index, endIndex));
          index = endIndex+1; */
          array[i] = Integer.parseInt(st.nextToken());
        }
//      String fileName = str.substring(index);
        String fileName = st.nextToken();
        getCurrentDate();

        if (isBetween(array)) {
          // start playing
          if (isAddress(fileName)) flag = false;
          else flag = true;

          System.out.println("fileName: "+fileName);
          if (flag)
            requestPkt = marconi.encodeRequest((byte)4, null, lmAddress, fileName);
          else
            requestPkt = marconi.encodeRequest((byte)1, fileName, lmAddress, null);

        marconi.sendRequest(requestPkt);

/*      System.out.println ("now in date range:");
        for (int i=0; i<8; i++) {
          System.out.print(array[i] + " ");
        }
        System.out.println();
        if (flag)
          System.out.println("Playing file "+fileName);
        else
          System.out.println("Redirecting packets from "+fileName); */
```

```
        while (true) {
          Thread.sleep(1000);
          getCurrentDate();
          if (!isBetween(array)) break;
        }
        // stop playing
          requestPkt = marconi.encodeRequest((byte)2, null, lmAddress, null);
          marconi.sendRequest(requestPkt);
        } // if
      }
    } //while
  }
  catch (Exception e) {
    e.printStackTrace();
  }
}

public boolean isAddress (String str) {
  int count=0;
  int index=0;
  while ((index=str.indexOf('.', index)) != -1) {
    count++;
    index++;
  }
  if (count == 3) return true;
  else return false;
}

public static void main(String args[]) throws Exception {
  try {
  if (args.length != 1) {
    System.out.println("Usage : Schedule <schedule data file>");
    return;
  }
  String scheduleFile = args[0];
  while (true) {
    BufferedReader br = new BufferedReader(new FileReader(scheduleFile));

    int[] array = new int[8];
    String str;
    while ((str = br.readLine()) != null) {
      int index = 0;
    for (int i=0; i<8; i++) {
      int endIndex = str.indexOf(' ', index);
      array[i] = Integer.parseInt(str.substring(index, endIndex));
      index = endIndex+1;
    }
    String fileName = str.substring(index);
    String localName = "LOCAL";
    getCurrentDate();

    if (isBetween(array)) {
      // PLAY
      System.out.println ("current time:");
      for (int i=0; i<4; i++) {
        System.out.print(dateArray[i] + " ");
      }
      System.out.println();
      System.out.println ("date range:");
      for (int i=0; i<8; i++) {
        System.out.print(array[i] + " ");
      }
      System.out.println();
      System.out.println("Playing file "+fileName);
```

```
        while (true) {
          Thread.sleep(1000);
          getCurrentDate();
          if (!isBetween(array)) break;
        }
      // STOP PLAYING
    } // if
    }
  } //while
  }
  catch (Exception e) {
    e.printStackTrace();
  }
} // main

public static void getCurrentDate() {
  Date myDate = new Date();
  Calendar myCalendar = Calendar.getInstance();
  dateArray[0] = myCalendar.get(Calendar.DAY_OF_WEEK)-1;
  dateArray[1] = myCalendar.get(Calendar.HOUR_OF_DAY);
  dateArray[2] = myCalendar.get(Calendar.MINUTE);
  dateArray[3] = myCalendar.get(Calendar.SECOND);
}

public static boolean isBetween(int[] array) {

  for (int i=array[0]; ;i=(i+1)%7) {
    if (i == dateArray[0]) {
      if (i != array[0] && i != array[4])
        return true; //between 2 days

      if (i == array[0]) { //sometime on the left boundary day
        if (dateArray[1] < array[1]) // if hour is earlier
          return false;
        if (dateArray[1] == array[1]) { // figure out the minutes
          if (dateArray[2] < array[2]) return false;
          if (dateArray[2] == array[2]) { // figure out the seconds
            if (dateArray[3] < array[3]) return false;
            if (dateArray[3] == array[3] || array[0] != array[4])
              return true;
          }
          else if (dateArray[0] != array[4]) return true;
        }
        else if (dateArray[0] != array[4]) return true;
      }
      if (i == array[4]) { //sometime on the right boundary day
        if (dateArray[1] > array[5]) // if time is later
          return false;
        else if (dateArray[1] < array[5]) //has already been checked for early
          return true;
        else { // hours are equal
          if (dateArray[2] < array[6]) return true;
          if (dateArray[2] > array[6]) return false;
          //minutes are equal
          if (dateArray[3] > array[7]) return false;
          return true;
        }
      }
      break;
    }
    if (i == array[4]) break;
  }
  return false;
}
```

```
} // class
```

NewEdit.java Thu Jun 17 14:19:03 1999 1

```
import java.awt.*;
import java.io.*;

public class NewEdit extends Frame {
  public NewEdit(String fileName, Frame f) {
    this.f = f;
    try {
      bw = new BufferedWriter(new FileWriter(fileName));
    }
    catch (IOException e) {
      System.exit(0);
    }
    this.fileName = fileName;
    setTitle("NewEdit");
    Panel p1 = new Panel();
    p1.setLayout(new FlowLayout());
    newButton = new Button ("New Entry");
    p1.add(newButton);
    finishButton = new Button ("Finish");
    p1.add(finishButton);
    add("Center", p1);
  }

  public void processEdit(EditInfo info) {
    String space = " ";
    try {
      bw.write(info.fromDay, 0, 1);
      bw.write(space, 0, 1);
      bw.write(info.fromHour, 0, info.fromHour.length());
      bw.write(space, 0, 1);
      bw.write(info.fromMin, 0, info.fromMin.length());
      bw.write(space, 0, 1);
      bw.write(info.fromSec, 0, info.fromSec.length());
      bw.write(space, 0, 1);
      bw.write(info.toDay, 0, 1);
      bw.write(space, 0, 1);
      bw.write(info.toHour, 0, info.toHour.length());
      bw.write(space, 0, 1);
      bw.write(info.toMin, 0, info.toMin.length());
      bw.write(space, 0, 1);
      bw.write(info.toSec, 0, info.toSec.length());
      bw.write(space, 0, 1);
      bw.write(info.filename, 0, info.filename.length());
      bw.newLine();
    }
    catch (IOException e) {
      System.exit(0);
    }
  }

  public boolean handleEvent(Event event) {
    if (event.id == Event.ACTION_EVENT && event.target == newButton) {
      EditInfo in = new EditInfo();
      EditDialog ed = new EditDialog(this, in);
      ed.show();
    }
    else if (event.id == Event.ACTION_EVENT && event.target == finishButton) {
      try {
        bw.close();
      }
      catch (IOException e) {

      }
      dispose();
```

```
      f.show();
    }
    return true;
  }

  private Button newButton, finishButton;
  private String fileName;
  private BufferedWriter bw;
  private Frame f;
}

class EditInfo {
  String fromDay, toDay, fromHour, toHour, fromMin, toMin, fromSec,
      toSec, filename;
  EditInfo(String fromDay, String toDay, String fromHour, String toHour,
      String fromMin, String toMin, String fromSec, String toSec,
      String filename) {
    this.fromDay = fromDay;
    this.toDay = toDay;
    this.fromHour = fromHour;
    this.toHour = toHour;
    this.fromMin = fromMin;
    this.toMin = toMin;
    this.fromSec = fromSec;
    this.toSec = toSec;
    this.filename = filename;
  }
  EditInfo() {}
}

class Days {
  List days;
  Days() {
    days = new List(1, false);
    days.addItem("0:Sun");
    days.addItem("1:Mon");
    days.addItem("2:Tue");
    days.addItem("3:Wed");
    days.addItem("4:Thu");
    days.addItem("5:Fri");
    days.addItem("6:Sat");
    days.select(0);
  }
}

class EditDialog extends Dialog {
  public EditDialog(NewEdit parent, EditInfo u) {
    super(parent, "Edit Entry", true);
    fromDay = new Days();
    toDay = new Days();
    Panel p1 = new Panel();
    p1.setLayout(new GridLayout(9, 2));
    p1.add(new Label("From day:"));
    p1.add(fromDay.days);
    p1.add(new Label("From hour:"));
    p1.add(fromHour = new TextField(2));
    p1.add(new Label("From minute:"));
    p1.add(fromMin = new TextField(2));
    p1.add(new Label("From second:"));
    p1.add(fromSec = new TextField(2));
    p1.add(new Label("To day:"));
    p1.add(toDay.days);
    p1.add(new Label("To hour:"));
    p1.add(toHour = new TextField(2));
```

```
    p1.add(new Label("To minute:"));
    p1.add(toMin = new TextField(2));
    p1.add(new Label("To second:"));
    p1.add(toSec = new TextField(2));
    p1.add(new Label("File/Channel:"));
    p1.add(filename = new TextField("", 20));
    CheckboxGroup g = new CheckboxGroup();
    Panel p3 = new Panel();
    p3.setLayout(new FlowLayout());
    p3.add(localBox = new Checkbox("Local", g, true));
    p3.add(globalBox = new Checkbox("Global", g, false));
    add("Center", p3);
    add("North", p1);
    Panel p2 = new Panel();
    p2.add(okButton = new Button("Ok"));
    p2.add(cancelButton = new Button("Cancel"));
    add("South", p2);
    resize(200, 400);
  }

  public boolean action(Event event, Object arg) {
    if (arg.equals("Ok")) {
      if (fromHour.getText().equals ("") || toHour.getText().equals("")
        || fromMin.getText().equals("") || toMin.getText().equals("") ||
        fromSec.getText().equals("") || toSec.getText().equals("")
        || filename.getText().equals(""))
        return true;
      try {
        if (Integer.parseInt(fromHour.getText()) < 0 ||
              Integer.parseInt(fromHour.getText()) > 23 ||
              Integer.parseInt(toHour.getText()) < 0 ||
              Integer.parseInt(toHour.getText()) > 23 ||
              Integer.parseInt(fromMin.getText()) < 0 ||
              Integer.parseInt(fromMin.getText()) > 59 ||
              Integer.parseInt(toMin.getText()) < 0 ||
              Integer.parseInt(toMin.getText()) > 59 ||
              Integer.parseInt(fromSec.getText()) < 0 ||
              Integer.parseInt(fromSec.getText()) > 59 ||
              Integer.parseInt(toSec.getText()) < 0 ||
              Integer.parseInt(toSec.getText()) > 59)
          return true;
        }
        catch (NumberFormatException e) {
          return true;
        }
        String fname = new String();
        String str;
        boolean flag = false;
        if (globalBox.getState()) {
          try {
            BufferedReader b = new BufferedReader(new FileReader("_mapsta"));
            while ((str = b.readLine()) != null) {
              int index = 0;
              int endIndex = str.indexOf(' ', index);
              String addr = str.substring(index, endIndex);
              System.out.println("str: "+str.substring(endIndex+1));
              if ((str.substring(endIndex+1)).compareTo(filename.getText())==0) {
                fname = addr;
                flag = true;
                break;
              }
            }
            b.close();
          }
```

```
        catch (IOException e) {}

        if (!flag) {
          System.out.println("Channel name "+filename.getText()+" not found");
          return true;
        }
      }
      else fname = filename.getText();
      dispose();

      EditInfo result = new EditInfo (fromDay.days.getSelectedItem(), toDay.days.getSelected
+ tem(),
        fromHour.getText(), toHour.getText(), fromMin.getText(),
        toMin.getText(), fromSec.getText(), toSec.getText(), fname);
       ((NewEdit)getParent()).processEdit(result);
    }
    else if (arg.equals("Cancel")) {
      dispose();
    }
    else return super.action(event, arg);
    return true;
  }

  public boolean handleEvent(Event evt) {
    if (evt.id == Event.WINDOW_DESTROY) dispose();
    else return super.handleEvent(evt);
    return true;
  }

  private TextField fromHour, toHour, fromMin, toMin, fromSec, toSec, filename;
  private Button okButton, cancelButton;
  private Days fromDay, toDay;
  private Checkbox localBox, globalBox;
}
```

IRCActionHandler.java Thu Jun 17 14:21:09 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [IRC Action Handler]
 *
 * $<marconi.irc.>IRCActionHandler.java -v2.0(prototype version), 1999/04/21 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * This class handles actions taken by IRC user.
 */
public class IRCActionHandler {
    public static ActionListener listControl = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            IRCControls.entry_1.setText
                (IRCDirectory.directoryList_1.getItem
                (IRCDirectory.directoryList_1.getSelectedIndex()));
            String textfield = IRCControls.entry_1.getText();
            StringTokenizer dir = new StringTokenizer(textfield, " ");
            int id = Integer.parseInt(dir.nextToken());
            if (IRCDirectory.owner.playChannel(id)) {
                System.out.println("marconi.irc.IRCActionHandler" +
                                   ".actionPerformed: playing channel " +
                                   + id + ".");
            }
            else {
                IRCDirectory.owner.stopChannel();
                System.err.println("marconi.irc.IRCActionHandler" +
                                   ".actionPerformed: error listening.");
            }
        }
    };
}
```

IRCControls.java Thu[illegible]n 17 14:21:52 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [IRC GUI Controls]
 *
 * $<marconi.irc.>IRCControls.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, ~riK.
 */
package marconi.irc;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.io.*;

/**
 * This panel contains user interfaces to the applet.
 */
public class IRCControls extends Panel {
    public static TextField entry_1;
    private static int WIDTH  = 600;
    private static int HEIGHT = 100;

    /**
     * Instantiates the control panel.
     */
    public IRCControls() {
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.NONE;
        cons.weightx = 0.0;

        // selected station (id + name)
        entry_1 = new TextField(40);
        grid.setConstraints(entry_1, cons);
        entry_1.setForeground(Color.yellow.darker());
        entry_1.setBackground(Color.blue.darker().darker());
        add(entry_1);
        validate();

        // resize
        setSize(WIDTH, HEIGHT);
    }
}
```

IRCDirectory.java Thu Jan 1 14:22:06 1999 1

```
/*
 * marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : (IRC Directory Controls)
 *
 * $<marconi.ras.>IRCDirectory.java -v2.0(prototype version), 1999/04/20 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.awt.*;
import java.awt.event.*;
import java.util.Hashtable;
import java.net.*;
import java.io.*;
import marconi.ras.MarconiServer;
import marconi.util.CDPPacket;

/**
 * This panel contains user interfaces to the applet.
 */
public class IRCDirectory extends Panel
    implements Runnable {

    /**
     * This thread updates the announcements for the locally supported channels.
     */
    private Thread updateThread = null;
    private static long L_UPDATE = 7500;

    public static IRCUsrApplet owner;
    public static Label label_1;
    public static List directoryList_1;
    private static int WIDTH  = 600;
    private static int HEIGHT = 400;

    /**
     * Instantiates the directory display panel.
     */
    public IRCDirectory(IRCUsrApplet main) {
        owner = main;
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.NONE;
        cons.weightx = 1.0;

        // label 1
        cons.weightx = 1.0;
        cons.gridwidth = GridBagConstraints.REMAINDER;
        label_1 = new Label();
        label_1.setText("Local Channel Directory");
        grid.setConstraints(label_1, cons);
        label_1.setForeground(Color.white);
        add(label_1);
        validate();

        // list 1 - global
        cons.gridwidth = GridBagConstraints.REMAINDER;
        directoryList_1 = new List(20, false);
        directoryList_1.addActionListener(IRCActionHandler.listControl);
        grid.setConstraints(directoryList_1, cons);
        directoryList_1.setForeground(Color.yellow.brighter());
        directoryList_1.setBackground(Color.darkGray);
        add(directoryList_1);
        validate();
```

IRCDirectory.java Thu Jun 17 14:22:06 1999 2

```
        // resize
        setSize(WIDTH, HEIGHT);
        start();
    }

    /**
     * Starts the directory update.
     */
    public void start() {
        updateThread = new Thread(this);
        updateThread.start();
    }

    /**
     * The run methods for the two threads.
     */
    public void run() {

        // local directory
        while (Thread.currentThread() == updateThread) {
            try {
                Thread.sleep(L_UPDATE);
                Hashtable cdp_lookup = owner.getCache();

                if (directoryList_1.getItemCount() > 0) {
                    directoryList_1.removeAll();
                }
                for (int i = 0; i < owner.MAX_CHANNELS; i++) {
                    String Id = String.valueOf(i);
                    if (cdp_lookup.containsKey(Id)) {
                        CDPPacket cdp = (CDPPacket) cdp_lookup.get(Id);
                        directoryList_1.add(cdp.id + " " + cdp.name,
                                            Integer.parseInt(cdp.id));
                    }
                    else {
                        directoryList_1.add(Id);
                    }
                }
                if (cdp_lookup.containsKey(MarconiServer.LOCALSTA)) {
                    CDPPacket cdp = (CDPPacket) cdp_lookup.get(MarconiServer.LOCALSTA);
                    directoryList_1.add(cdp.name);
                }
            }
            catch (Exception e) {
                System.err.println("marconi.irc.IRCDirectory.run:");
                e.printStackTrace();
            }
        }
    }
}
```

IRCUsrApplet.java Tue Jun 7 14:22:23 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : (IRC User Applet)
 *
 * $<marconi.irc.>IRCUsrApplet.java -v2.0(prototype version), 1999/04/20 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.applet.*;
import java.awt.*;
import java.util.*;
import java.net.*;
//import java.rmi.*;
//import java.rmi.server.*;
import marconi.util.*;
import marconi.util.rtsp.IRC;
import marconi.ras.RAS;
import marconi.ras.MarconiServer;

/**
 * This applet is used by an IRC user.
 *
 * @author -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.MarconiServer
 * @since   prototype v1.0
 */
public class IRCUsrApplet extends Applet
    implements java.io.Serializable, Runnable {

    /**
     * Session Announcement Protocol (SAP) resources.
     * Interfaced via. Channel Directory/Description Protocol (CDP).
     */
    private MulticastSocket cap_receiver = null;
    protected Hashtable capCache = null;

    /**
     * This thread updates the announcements for the locally supported channels.
     */
    private Thread directoryThread = null;
    private static long L_UPDATE = 10000;

    // miscellaneious variables
    private static int width = 0;
    private static int height = 0;
    //protected RAS rasServer = null;
    protected int MAX_CHANNELS = 20;
    final static String obj_name = "marconi.ras.IRCUsrApplet";

    // tools
    IRCDirectory directory = null;
    IRCControls controls = null;
    IRC listener = null;

    /**
     * Initialize the applet and setup display area.
     */
    public void init() {
        try {
            width = Integer.parseInt(getParameter("APPLWIDTH"));
            height = Integer.parseInt(getParameter("APPLHEIGHT"));

            // lookup RAS (MarconiServer)
```

```
            //URL URLBase = getDocumentBase();
            //System.out.println(obj_name + ".init: locating server");
            //rasServer = (RAS) Naming.lookup("//" + getParameter("RASHost")
            //                                + ":" + getParameter("RASPort")
            //                                + "/marconi.ras.MarconiServer");

            // init variables
            //MAX_CHANNELS = rasServer.getMaxChannels();
            capCache = new Hashtable();
        }
        catch (Exception e) {
            // fatal error
            System.err.println(obj_name + ".init: ");
            e.printStackTrace();
        }

        // join CAP multicast group
        try {
            cap_receiver = new MulticastSocket(MarconiServer.CAP_PORT);
            cap_receiver.joinGroup(InetAddress.getByName(MarconiServer.LOCAL_CAP));
        }
        catch (Exception e) {
            System.out.println(obj_name + ".init:");
            e.printStackTrace();
        }

        // draw display area
        setupDisplay();

        // start
        listener = new IRC();
        directoryThread = new Thread(this);
        directoryThread.start();
    }

    /**
     * Display the applet.
     */
    public void setupDisplay() {
        setBackground(Color.black);

        directory = new IRCDirectory(this);
        controls = new IRCControls();
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();

        // setup grid
        int rowHeights[] = {400, 100};
        grid.rowHeights = rowHeights;
        setLayout(grid);
        cons.fill = GridBagConstraints.BOTH;

        // add directory lists
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
        grid.setConstraints(directory, cons);
        add(directory);
        validate();

        // add controls
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
```

```
        grid.setConstraints(controls, cons);
        add(controls);
        validate();

        resize(width, height);
    }

    /**
     * Free resources when closing applet.
     */
    public void destroy() {
        // release sockets and leave announcement group
        try {
            cap_receiver.leaveGroup(InetAddress.getByName(MarconiServer.LOCAL_CAP));
            cap_receiver.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        stopChannel();
        directoryThread = null;

        //try {
        //    rasServer.terminate();
        //}
        //catch (RemoteException e) {
        //    e.printStackTrace();
        //}

        remove(directory);
        remove(controls);
    }

    /**
     * Run method.
     */
    public void run() {

        // cache update hour
        long hour = System.currentTimeMillis();

        /*
        * Global directory thread running.
        */
        while (Thread.currentThread() == directoryThread) {

            /*
            * Receive CDP packets and maintain channel database.
            */
            try {
                DatagramPacket recv_pkt = CDPPacket.compose();
                cap_receiver.receive(recv_pkt);

                CDPPacket cdp = new CDPPacket(recv_pkt);
                System.out.println(obj_name + ".run: cdp parsed");

                // update announcement appropriately
                if (!capCache.containsKey(cdp.id)) {
                    capCache.put(cdp.id, cdp);
                    System.out.println(obj_name + ".run: new cdp cached for channel-"
                                       + cdp.id);
                }
            }
            catch (Exception e) {
```

IRCUsrApplet.java Thu Jun 17 14:22:23 1999 4

```
                e.printStackTrace();
            }

            // time to refresh cache (daily)
            long current_hour = System.currentTimeMillis();
            if (current_hour > hour + Timestamp.DAY) {
                System.out.println(obj_name + ".run: routine -refreshing directory cache.");
                refresh_cache();
                hour += Timestamp.DAY;
            }

            /*
            * Interval b/w each loop for receiving local channel directory.
            */
            try {
                Thread.sleep(L_UPDATE);
            }
            catch (InterruptedException e) {
            }
        }
    }

    /**
     * Removes old cache entries.
     */
    protected void refresh_cache() {
        long current_hour = System.currentTimeMillis();

        synchronized (capCache) {
            Enumeration capList = capCache.elements();
            while (capList.hasMoreElements()) {
                CDPPacket cdp = (CDPPacket) capList.nextElement();
                if (current_hour > cdp.timeStamp + CDPPacket.TTL) {
                    capCache.remove(cdp.id);
                }
            }
        }
    }

    /**
     * Inform server that the specified channel is being listen to. This
     * RMI based triggering is very inefficient and not scalable. So
     * alternate approach based on RTCP should replace this.
     */
    public boolean playChannel(int id) {
        boolean status = false;

        if (capCache.containsKey(String.valueOf(id))) {
            CDPPacket cdp = (CDPPacket) capCache.get(String.valueOf(id));
            try {
                // kill previous thread if running
                listener.stop();
                /*
                * The below statement is commented out because the listener
                * is now capable of sending RTCP signals for triggering.
                *
                status = rasServer.playChannel(id);
                */
                status = true; // replace above
                listener.start(cdp.MADDR, cdp.MPORT);
            }
            catch (Exception e) {
                e.printStackTrace();
                return false;
```

```
            }
            return status;
        }
        else {
            return false;
        }
    }

    /**
     * Stops listening to whatever is playing.
     */
    public void stopChannel() {
        listener.stop();
    }

    /**
     * Returns the current state of the local channel announcement cache.
     */
    protected synchronized Hashtable getCache() {
        return capCache;
    }

    /**
     * Return applet information.
     */
    public String getAppletInfo() {
        return "IRC listener tool";
    }
}
```

ADControls.java Thu [redacted] 17 14:23:40 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [AD GUI Controls]
 *
 * $<marconi.ras.>ADControls.java -v2.0(prototype version), 1999/05/16 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import marconi.util.Vector2;

/**
 * This panel contains user interfaces for the ad insertion.
 */
public class ADControls extends Panel {
    public static RASMgrApplet owner;
    public static Choice ids;
    public static TextField entry_1;
    public static Button button_2, button_3, button_4;
    public static List directoryList_1;
    private static int WIDTH  = 600;
    private static int HEIGHT = 200;
    private static Vector2 ads = new Vector2();

    /**
     * Instantiates the control panel.
     */
    public ADControls(RASMgrApplet main) {
        owner = main;

        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.BOTH;
        cons.weightx = 0.0;

        // channel id
        ids = new Choice();
        updateIDs();
        grid.setConstraints(ids, cons);
        ids.setForeground(Color.black);
        //        ids.setForeground(Color.yellow.darker());
        ids.setBackground(Color.lightGray);
        //        ids.setBackground(Color.black);
        add(ids);
        validate();

        // commercial file input
        entry_1 = new TextField(20);
        grid.setConstraints(entry_1, cons);
        entry_1.setForeground(Color.black);
        //        entry_1.setForeground(Color.yellow.darker());
        entry_1.setBackground(Color.gray.brighter());
        //        entry_1.setBackground(Color.blue.darker().darker());
        add(entry_1);
        validate();

        // add commercial
        cons.gridwidth = GridBagConstraints.REMAINDER;
        button_2 = new Button(" Add Commercial ");
        button_2.setActionCommand("Add Commercial");
        button_2.addActionListener(RASActionHandler.buttonControl);
```

```
        button_2.setBackground(Color.lightGray);
        //        button_2.setBackground(Color.black);
        button_2.setForeground(Color.black);
        //        button_2.setForeground(Color.red);
        grid.setConstraints(button_2, cons);
        add(button_2);
        validate();

        // list
        //        cons.weightx = 1.0;
        cons.gridwidth = GridBagConstraints.REMAINDER;
        directoryList_1 = new List(5, false);
        grid.setConstraints(directoryList_1, cons);
        directoryList_1.setForeground(Color.black);
        //        directoryList_1.setForeground(Color.yellow.brighter());
        directoryList_1.setBackground(Color.white);
        //        directoryList_1.setBackground(Color.darkGray);
        add(directoryList_1);
        validate();

        // submit commercial list
        cons.gridwidth = GridBagConstraints.RELATIVE;
        button_3 = new Button(" Submit List ");
        button_3.setActionCommand("Submit Commercials");
        button_3.addActionListener(RASActionHandler.buttonControl);
        button_3.setBackground(Color.lightGray);
        //        button_3.setBackground(Color.black);
        button_3.setForeground(Color.black);
        //        button_3.setForeground(Color.red);
        grid.setConstraints(button_3, cons);
        add(button_3);
        validate();

        // remove commercial
        cons.gridwidth = GridBagConstraints.REMAINDER;
        button_4 = new Button(" Remove All ");
        button_4.setActionCommand("Remove Commercials");
        button_4.addActionListener(RASActionHandler.buttonControl);
        button_4.setBackground(Color.lightGray);
        //        button_4.setBackground(Color.black);
        button_4.setForeground(Color.black);
        //        button_4.setForeground(Color.red);
        grid.setConstraints(button_4, cons);
        add(button_4);
        validate();

        // resize
        setSize(WIDTH, HEIGHT);
    }

    /**
     * Add additional entry.
     */
    protected static void processAdd() {
        if (entry_1.getText().length() > 0) {
            try {
                String entry = Integer.parseInt(ids.getSelectedItem()) + " " + entry_1.get
+ t();
                directoryList_1.add(entry);
                ads.addElement(entry);
            }
            catch (NumberFormatException e) {
                owner.showMessage("Please select a channel!", true);
            }
```

```
        }

        // clear input fields
        entry_1.setText("");
    }

    /**
     * Remove all entries.
     */
    protected static void processRemove() {
        if (ads.size() > 0) {
            directoryList_1.removeAll();
            ads.removeAllElements();
        }
    }

    /**
     * Submit the commercial list to the server.
     */
    protected static void processSubmit() {
        try {
            if (ads.size() > 0 && owner.rasServer.submitCommercialList(ads.toStringArray()))
{
                owner.showMessage("Commercial list submitted...", false);
                processRemove();
                entry_1.setText("");
            }
            else {
                owner.showMessage("Commercial list could not be submitted!", true);
            }
        }
        catch (Exception e) {
            owner.showMessage("Commercial list could not be submitted!", true);
        }
    }

    /**
     * Updates available channel ids.
     */
    protected static void updateIDs() {
        ids.removeAll();

        for (int i = 0; i < owner.channelIDs.length; i++) {
            if (owner.channelIDs[i]) {
                ids.add(String.valueOf(i));
            }
        }
        if (ids.getItemCount() == 0) {
            ids.add("EMPTY");
        }
    }
}
```

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Channel Class]
 *
 * $<marconi.ras.>Channel.java -v2.0(prototype version), 1999/01/06 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.io.*;
import java.net.*;
import java.util.*;
import java.rmi.*;
import marconi.util.*;
import marconi.util.rtsp.*;
import marconi.rsc.RSC;
import java.security.Security;
import javax.crypto.*;
import javax.crypto.spec.*;
import au.net.aba.crypto.provider.ABAProvider;

/**
 * The <code>Channel</code> class creates two threads in which it can start
 * the neccessary operations for maintaining the channel. First, it creates
 * a thread that listens and monitors the RTCP signals from IRCs. When this
 * thread detects that there is at least one IRC that wants to listen to this
 * channel, it creates the second thread which begins performing the redirection
 * process (receive content from global address, decrypt, write content to
 * local address). The <i>rtcp thread</i> continues the monitoring so that
 * if it detects that no one is listening to the channel anymore it terminates
 * the <i>redirection thread</i>. This helps reduce the bandwidth being wasted.
 * Additionally, <code>Channel</code> will store the details of its content
 * provider (radio station) and the broadcast/multicast medium.
 * <p>
 *
 * @author  ~riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.MarconiServer
 * @see     marconi.ras.StationProfile
 * @since   prototype v1.0
 */
public class Channel implements Runnable {
    final static String obj_name = "marconi.ras.Channel";

    /**
     * The assigned channel number.
     */
    protected int chanID;

    /**
     * The radio station.
     */
    private RSC station = null;

    /**
     * The radio station's hostname.
     */
    private String host = null;

    /**
     * The radio station name.
     */
    public String name = null;

    /**
```

```
     * The main category of this channel's content (music|news|sports|...).
     */
    public String category = null;

    /**
     * The content description.
     */
    public String description = null;

    /**
     * The origin of the content.
     */
    public String origin = null;

    /**
     * The language used in the content.
     */
    public String language = null;

    /**
     * The radio station's url (not in use by the current protocol).
     */
    public URL url = null;

    /**
     * The station's glocal multicast address.
     */
    protected InetAddress g_maddr;

    /**
     * The station's local multicast address.
     */
    protected InetAddress l_maddr;

    /*
    * The encryption resources. These private declaration allows for later
    * extension where the hard-coded values such as the SEK algorithm and key
    * length can be obtained from announcement. In such cases various encryption
    * methods can be supported and allows the encrypting party (content sender) to
    * decide on which algorithm to use.
    */
    private byte[] SEK = null;                          // session encryption key
    int sek_id = -1;                                    // RSC registration id
    byte[] publicKey = null;                            // RSA public key
    byte[] privateKey = null;                           // RSA private key
    private final static String SECRET_ALG = "RC4";     // encryption algorithm
    private final static int SEKLEN = 8;                // encryption key length

    /**
     * Channel database initialized (install JCE and thread counter).
     */
    private static boolean DB_INITIALIZED = false;

    /**
     * Daily program schdule.
     */
    private Hashtable programSchedule = null;

    /**
     * Daily commercial schedule.
     */
    private Hashtable commercialSchedule = null;

    /**
```

```
     * The radio antenna server can activate the channel broadcasting by
     * starting this <code>channelThread</code>.
     */
    private Thread channelThread = null;

    /**
     * The channel broadcasting does not actually begin unless this RTCP listener
     * thread detects that there is at least one listener present at the time.
     */
    private Thread rtcpThread = null;

    /**
     * This variable indicates the state of the thread (active or suspended).
     */
    private boolean threadSuspended = false;

    /**
     * Manages the number of running threads (channels).
     */
    protected static ThreadCountManager threadCounter = null;

    /**
     * The RTCP listener.
     */
    private RTCPListener rtcpListener = null;
    private RTCPSocket rtcpRegistry = null;

    /*
    * Multicast resources.
    */
    byte recv_buf[];                    // receiver buffer
    byte send_buf[];                    // receiver buffer
    DatagramPacket recv_pack;           // receiver packet
    DatagramPacket send_pack;           // receiver packet
    private MulticastSocket recv_msock; // multicast socket for receiving
    private MulticastSocket send_msock; // multicast socket for sending
    private boolean SOCKET_IN_USE = false; // prevents abrupt socket kill

    /**
     * Creates a new thread and starts the <code>Channel</code> operation
     * which consists of broadcasting (multicasting) the channel contents
     * to its local clients and storing the station profile.
     *
     * @param   chan_id the channel number.
     * @param   maddr   the local multicast address.
     */
    public Channel(int chan_id, InetAddress maddr) {

        // install jce provider
        if (!DB_INITIALIZED) {
            threadCounter = new ThreadCountManager();
            DB_INITIALIZED = Security.addProvider(new ABAProvider()) > -1;
        }

        // initialize channel
        chanID = chan_id;
        programSchedule = new Hashtable();
        commercialSchedule = new Hashtable();
        g_maddr = null;
        l_maddr = maddr;

        // initialize rtcp listener
        rtcpRegistry = new RTCPSocket(MaddrDispenser.rtcp_map(maddr),
                                      MarconiServer.IRC_PORT+1, true);
```

```
        rtcpListener = new RTCPListener(rtcpRegistry,
        rtcpListener.start();
        rtcpThread = new Thread(this);
        rtcpThread.start();
    }

    /**
     * This <code>run</code> method implements the channel's broadcast
     * operation.
     */
    public void run() {

        /*
        * Channel thread running.
        */
        while (Thread.currentThread() == channelThread) {

            // receive audio packet
            SOCKET_IN_USE = true;
            try {
                // init receiving packet
                recv_buf = new byte[MarconiServer.MAX_PAYLOADLEN
                                + MarconiServer.MAX_RTPHDRLEN];
                recv_pack = new DatagramPacket(recv_buf, recv_buf.length);
                recv_msock.receive(recv_pack);

                //System.out.println("received RTP from " + recv_pack.getAddress() + ":" + r
+ cv_pack.getPort());
            }
            catch (Exception e) {
                System.out.println(obj_name + ".run: channel-" + chanID);
                e.printStackTrace();
            }
            SOCKET_IN_USE = false;

            // Decrypt & send local
            int n = recv_pack.getLength();
            if (n > 0 && SEK != null) {
                try {
                    //Cipher cipher = Cipher.getInstance(SECRET_ALG);
                    //cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(SEK, SECRET_ALG));
                    //byte[] send_buf = cipher.doFinal(recv_pack.getData());
                    byte[] send_buf = recv_pack.getData();
                    send_pack = new DatagramPacket(send_buf, n, l_maddr,
                                                   MarconiServer.IRC_PORT);
                    send_msock.send(send_pack, (byte) MarconiServer.LOCAL_TTL);
                    //System.out.println("sent to " + send_pack.getAddress() + ":" + send_p
+ k.getPort());
                }
                catch (Exception e) {
                    System.out.println(obj_name + ".run: ");
                    e.printStackTrace();
                }
            }

            /* Uncomment below to use variable sleep feature. (and comment above).
            * Sleep. (accordingly wrt # of running channels)
            */
            try {
                Thread.sleep(threadCounter.getSleepTime());

                synchronized (this) {
                    while (threadSuspended && channelThread != null ) {
                        wait();
```

```
                    }
                }
            }
            catch (InterruptedException e) {
            }
            /**/
            // let other channels (threads) run (time-slicing)
            //Thread.yield();
        }

        /*
        * RTCP listener thread running.
        */
        while (Thread.currentThread() == rtcpThread) {

            //System.out.println(obj_name + ".run: channel-" + chanID + " num ircs=" + rtcpRe
gistry.getNumMembers());
            // determine whether or not to start the channel thread
            if (!isOnline() && rtcpRegistry.getNumMembers() > 0) {
                try {
                    start();
                    System.out.println(obj_name + ".run: channel-" + chanID
                                       + " started");
                }
                catch (Exception e) {
                    System.err.println(obj_name +
                                       ".run: cannot start channel-" + chanID);
                }
            }

            // determine whether or not to stop the channel thread
            if (isOnline() && rtcpRegistry.getNumMembers() == 0) {
                stop();
                System.out.println(obj_name + ".run: channel-" + chanID
                                   + " stopped");
            }
            /* Uncomment below to use variable sleep feature. (and comment above).
            * Sleep. (accordingly wrt # of running channels)
            */
            try {
                Thread.sleep(threadCounter.getSleepTime());

                synchronized (this) {
                    while (threadSuspended && channelThread != null ) {
                        wait();
                    }
                }
            }
            catch (InterruptedException e) {
            }
            /**/
            // let other channels (threads) run (time-slicing)
            //Thread.yield();
        }
    }

    /**
     * Starts the broadcast thread for this channel. The channel will begin
     * to listen on its content provider's data stream and do its protocol
     * process (strip-off header, check for special packet, decrypt, etc.)
     * and broadcast (multicast) the remaining audio data to all clients tuned
     * to this channel.
     *
     * @exception  TooManyChannelThreadsException if too many
```

```
     *                <code>channelThread</code>s are running (broadcasting).
     */
    public synchronized void start() throws TooManyChannelThreadsException {

        if (!isOnline()) {
            if (g_maddr != null && threadCounter.addThread()) {
                System.out.println(obj_name
                                   + ".start: initializing channel-"
                                   + chanID);

                // create sockets & join group
                try {
                    send_msock = new MulticastSocket();
                    recv_msock = new MulticastSocket(MarconiServer.RSC_PORT);
                    recv_msock.joinGroup(g_maddr);
                }
                catch (IOException e) {
                    System.err.println(obj_name + ".start:");
                    e.printStackTrace();
                }

                // start channel announcement thread
                channelThread = new Thread(this);
                channelThread.start();
            }
            else {
                throw new TooManyChannelThreadsException
                    (obj_name + ".start: too many threads running");
            }
        }
        else {
            throw new IllegalThreadStateException
                (obj_name
                + ".start: broadcast thread already running for channel-"
                + chanID);
        }
    }

    /**
     * Stops the broadcast thread for the channel. The channel itself is not
     * to be destroyed.
     */
    public synchronized void stop() {

        if (isOnline()) {
            if (threadCounter.removeThread()) {

                // free sockets & threads
                try {
                    channelThread = null;
                    while (SOCKET_IN_USE) {
                        Thread.sleep(3);
                    }
                    recv_msock.leaveGroup(g_maddr);
                    recv_msock.close();
                    send_msock.close();
                }
                catch (Exception e) {
                    System.err.println(obj_name + ".stop:");
                    e.printStackTrace();
                }
                finally {
                    notify();
                }
```

```
            }
            else {
                // do nothing
            }
        }
        else {
            throw new IllegalThreadStateException
                (obj_name
                + ".stop: broadcast thread not running for channel-"
                + chanID);
        }
    }

    /**
     * Stop all threads and destroy this object.
     */
    public synchronized void destroy() {
        stop();
        rtcpThread = null;
        rtcpListener.stop();
        rtcpRegistry.close();
    }

    /**
     * Temporarily suspends the running broadcast thread.
     */
    public synchronized void suspend() {

        if (isOnline()) {
            threadSuspended = true;
            notify();
        }
        else {
            throw new IllegalThreadStateException
                (obj_name
                + ".suspend: broadcast thread not running for channel-"
                + chanID);
        }
    }

    /**
     * Resumes the broadcast operation.
     */
    public synchronized void resume() {

        if (isOnline()) {
            threadSuspended = false;
            notify();
        }
        else {
            throw new IllegalThreadStateException
                (obj_name
                + ": broadcast thread not running for channel-"
                + chanID);
        }
    }

    /**
     * Tests to see if thread is running (i.e. channel is broadcasting).
     */
    protected synchronized boolean isOnline() {
        return (channelThread != null) ? true : false;
    }
```

```
    /**
     * Initializes broadcasting process. Registers (make payment) with the radio
     * station to obtain the SEK. Set key pair (public/private).
     *
     * @param   pubKey  public key of RAS, as given by <code>MarconiServer</code>.
     * @param   privKey private key of RAS.
     */
    protected synchronized boolean init(byte[] pubKey, byte[] privKey) {
        this.publicKey = pubKey;
        this.privateKey = privKey;
        try {
            System.out.println(obj_name + ".init: contacting RSC at "
                               + host + "...");
            station = (RSC) Naming.lookup("//" + this.host
                                          + ":" + MarconiServer.RMI_PORT
                                          + "/marconi.rsc.station");
            sek_id = station.enroll(publicKey);
            System.out.println(obj_name
                               + ".init: public key submitted (sekid=" + sek_id + ")");
        }
        catch (Exception e) {
            return false;
        }
        if (sek_id > -1) {
            return true;
        }
        else {
            return false;
        }
    }

    /**
     * Adds individual commercial into the channel database.
     *
     * @param   ad_id   advertisement id or commercial filename.
     */
    protected synchronized void addCommercial(String ad_id) {

        // set date using the time at GMT at 0 hour (midnight)
        Date date = new Date(Timestamp.get_midnight());

        if (!commercialSchedule.containsKey(date)) {
            Vector2 ad_list = new Vector2();
            commercialSchedule.put(date, ad_list);
        }
        Vector2 ad_list = (Vector2) commercialSchedule.get(date);
        ad_list.addElement(ad_id);
    }

    /**
     * Removes the list of commercials in the database.
     */
    protected synchronized void removeCommercials() {

        // set date using the time at GMT at 0 hour (midnight)
        Date date = new Date(Timestamp.get_midnight());

        if (commercialSchedule.containsKey(date)) {
            Vector2 ad_list = (Vector2) commercialSchedule.get(date);
            ad_list.removeAllElements();
        }
    }

    /**
```

```
     * Generates a commercial-list file from the channel dat
     * of a list of filenames to be read-in by the advertisement insertion module. Each
     * file in the list should contain the actual audio data for the commercial playing
     * and its name is added to the database via. <code>addCommercial()</code> method.
     * The commercial-list file takes the global multicast address of this station in
     * numerical format appended by ".lst".
     */
    protected synchronized void genCommercialFile() {
        Date date = new Date(Timestamp.get_midnight());

        if (commercialSchedule.containsKey(date)) {
            try {
                File file = new File(g_maddr.getHostAddress() + ".lst");
                PrintWriter fout = new PrintWriter(new BufferedWriter(new FileWriter(file)));

                Vector2 ad_list = (Vector2) commercialSchedule.get(date);
                String[] ad_arr = ad_list.toStringArray();
                if (ad_arr != null) {
                    for (int i = 0; i < ad_arr.length; i++) {
                        fout.println((ad_arr[i] != null) ? ad_arr[i] : "");
                    }
                }
                System.out.println(obj_name + ".genCommercialFile: file updated -"
                                   + new Date());
                fout.close();
            }
            catch (IOException e) {
            }
        }
    }

    /**
     * Returns channel statistics.
     *
     * @return  channel accounting information packaged in <code>ChannelStatistics</code>
     *          class.
     */
    protected synchronized ChannelStatistics audit() {
        return new ChannelStatistics(String.valueOf(chanID), rtcpRegistry.getNumMembers());
    }

    /**
     * Reads from a CDP packet and extracts channel information.
     *
     * @param   cdp the channel description packet to be parsed.
     */
    protected synchronized void read_cdp(CDPPacket cdp) {
        if (cdp.name != null) {
            this.name = cdp.name;
        }
        if (cdp.category != null) {
            this.category = cdp.category;
        }
        if (cdp.description != null) {
            this.description = cdp.description;
        }
        if (cdp.language != null) {
            this.language = cdp.language;
        }
        if (cdp.origin != null) {
            this.origin = cdp.origin;
        }
        if (cdp.MADDR != null) {
            try {
```

```
                this.g_maddr = InetAddress.getByName(cdp.MADDR);
            }
            catch (Exception e) {
            }
        }
        if (cdp.id != null) {
            this.host = cdp.id;
        }
        if (cdp.SEKLIST != null) {
            if (sek_id >= 0 && sek_id < cdp.SEKLIST.length) {
                if (cdp.SEKLIST[sek_id] != null) {
                    SEK = new byte[SEKLEN];
                    byte[] temp_byte = Base64.decode(cdp.SEKLIST[sek_id].getBytes());
                    System.arraycopy(temp_byte, 0, SEK, 0,
                                     Math.min(temp_byte.length, SEKLEN));
                }
            }
        }
        if (cdp.date != -1 && cdp.schedule != null) {

            // set date using the time at GMT at 0 hour (midnight)
            Date date = new Date(Timestamp.get_midnight(cdp.date));

            // store schedule
            programSchedule.put(date, cdp.schedule);

            /*
            * The commercial database design is not utilized in this version.
            * Instead, refer to genCommercialFile() and its related methods for
            * supporting LAIP (local advertisement insertion protocol).
            *
            commercialSchedule.put(date, new CommercialSchedule());
            */
        }
    }

    /**
     * Writes current channel information to a CDP packet. The scheduling
     * information is only done for the current date. Thus the server is not
     * supported to post announcements for future dates.
     *
     * @return  a fully initialized CDP packet.
     */
    protected synchronized CDPPacket write_cdp() {
        CDPPacket cdp = new CDPPacket();

        // get today's date at 0 hour.
        Date today = new Date(Timestamp.get_midnight());

        // copy channel info
        cdp.name = this.name;
        cdp.category = this.category;
        cdp.id = String.valueOf(this.chanID);
        cdp.description = this.description;
        cdp.origin = this.origin;
        cdp.language = this.language;
        cdp.MADDR = this.l_maddr.getHostAddress();
        cdp.MPORT = MarconiServer.IRC_PORT;
        cdp.MTTL = MarconiServer.LOCAL_TTL;

        // copy schedule info
        cdp.date = today.getTime();
        String sched = (String) this.programSchedule.get(today);
        cdp.schedule = (sched == null) ? "" : sched;
```

```
        return cdp;
    }
}


/**
 * The <code>ThreadCountManager</code> class implements the operations done
 * on the number of running threads. It defines the maximum number of threads
 * (i.e. channels). It also calculates the announcement interval time by taking
 * number of channels into account.
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @since   prototype 1.0
 */
class ThreadCountManager {

    int threadCount = 0;
    public final static int MAX_THREADCOUNT = MarconiServer.MAX_CHANNELS;
    final static int MAX_SLEEPTIME = 200;
    final static int MIN_SLEEPTIME = 20;

    /**
     * Increments the thread counter.
     */
    protected boolean addThread() {
        if (threadCount < MAX_THREADCOUNT) {
            threadCount++;
            return true;
        }
        return false;
    }

    /**
     * Decrements the thread counter.
     */
    protected boolean removeThread() {
        if (threadCount > 0) {
            threadCount--;
            return true;
        }
        return false;
    }

    /**
     * Tells whether there are no threads.
     */
    protected boolean isEmpty() {
        return (threadCount == 0) ? true : false;
    }

    /**
     * Returns the number of running threads.
     */
    protected int getCount() {
        return threadCount;
    }

    /**
     * Returns each thread's appropriate sleep time in number of milliseconds.
     * The more threads running, the less sleep time for each thread.
     */
```

Channel.java Thu Jun 17 14:23:49 1999 File

```
    protected int getSleepTime() {
        return Math.max(MAX_SLEEPTIME - threadCount * 10, MIN_SLEEPTIME);
    }
}
```

ChannelMonitorApplet.java Thu Jun 17 14:24:02 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Channel Monitoring Applet]
 *
 * $<marconi.ras.>ChannelMonitorApplet.java -v2.0(prototype version), 1999/05/22 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.applet.Applet;
import java.awt.*;
import java.util.*;
import java.net.*;
import java.rmi.*;
import java.rmi.server.*;

/**
 * The ChannelMonitorApplet exports a remote object, and periodically contacts RAS
 * to obtain channel accounting information and plots it in graph format.
 */
public class ChannelMonitorApplet extends Applet
    implements Runnable {
    final static int INCR = 10;
    final static int GRIDLEFT = 150;
    private static int WIDTH  = 600;
    private static int HEIGHT = 350;

    private static String title = "";
    final static String obj_name = "marconi.ras.ChannelMonitorApplet";
    Thread updateThread = null;

    private Hashtable channelTable = new Hashtable();
    private RAS rasServer = null;

    /**
    * Updates channel status.
    */
    public void update() {
        ChannelStatistics[] channels = null;
        try {
            channels = rasServer.getStatistics();
            System.out.println(obj_name + ".update: " + new Date());
        }
        catch (RemoteException e) {
        }
        for (int i = 0; i < channels.length; i++) {
            if (channels[i] != null) {
                ChannelData data = (ChannelData) channelTable.get(channels[i].chan_id);
                if (data != null) {
                    data.update(channels[i]);
                }
            }
        }
        repaint();
    }

    /**
     * Periodically update.
     */
    public void run() {
        while (Thread.currentThread() == updateThread) {
            try {
                Thread.sleep(10000);
            }
            catch (InterruptedException e) {
```

```
            }
            update();
        }
    }

    public void start() {
        updateThread = new Thread(this);
        updateThread.start();
    }

    public void stop() {
        updateThread = null;
    }

    /**
    * Initializes the applet.
    */
    public void init() {
        try {
            // lookup RAS server
            URL URLbase = getDocumentBase();
            System.out.println(obj_name + ".init: looking up RAS");
            rasServer = (RAS) Naming.lookup("//" + URLbase.getHost() + ":"
                                            + getParameter("RASPort")
                                            + "/marconi.ras.MarconiServer");
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        ChannelStatistics[] channels = null;
        try {
            channels = rasServer.getStatistics();
            System.out.println(obj_name + ".init: " + new Date());
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }

        for (int i = 0; i < channels.length; i++) {
            if (channels[i] != null && !channelTable.containsKey(channels[i].chan_id)) {
                channelTable.put(channels[i].chan_id, new ChannelData(channels[i].chan_id));
            }
        }
        setBackground(Color.white);
        setLayout(null);

        // draw checkboxes
        int i = 0;
        Enumeration enum =  channelTable.elements();
        while (enum.hasMoreElements()) {
            ChannelData data = (ChannelData) enum.nextElement();
            SmartCheckbox cb = new SmartCheckbox (data, this);
            data.cb = cb;
            add(cb);
            validate();
            cb.setState(data.displayed);
            cb.setBounds(10, i++*30+25, 60, 18);
        }
    }

    /**
    * Called when applet is destroyed.
    */
```

ChannelMonitorApplet.java Thu Jun 17 14:24:02 1999

```
    public void destroy() {
    }

    /**
    * Paints the panel.
    */
    public void paint(Graphics g) {

        // draw title
        g.setColor(Color.black);
        g.drawString("Channels monitored:", 10, 10);

        // draw grid lines
        g.setColor(Color.darkGray);
        for (int i = GRIDLEFT; i < WIDTH; i += 50) {
            g.drawLine(i, 0, i, HEIGHT - 50);
        }
        for (int i = 0; i < HEIGHT; i += 50) {
            g.drawLine(GRIDLEFT, i, WIDTH - 50, i);
        }
        g.setColor(Color.black);
        for (int i = 0; i < HEIGHT; i += 50) {
            int x = i / 2;
            if (x >= 100) x = 24;
            else if (x >= 25) x = 17;
            else x = 10;
            g.drawString(String.valueOf(i/2), GRIDLEFT - x, HEIGHT - 50 - i);
        }

        // draw channels
        Enumeration enum = channelTable.elements();
        while (enum.hasMoreElements()) {
            ChannelData data = (ChannelData) enum.nextElement();

            int size;
            ChannelStatistics[] updates;
            synchronized (data.updates) {
                size = data.updates.size();
                updates = new ChannelStatistics[size];
                data.updates.copyInto(updates);
            }

            g.setColor(data.color);
            if (data.displayed) {
                // draw box around checkbox if mouse is over it
                if (data.cb != null && data.cb.haveMouse()) {
                    Point p = data.cb.getLocation();
                    Dimension d = data.cb.getSize();
                    g.drawRect(p.x-1, p.y-1, d.width+4, d.height+4);
                    g.drawRect(p.x-2, p.y-2, d.width+4, d.height+4);

                    // point to graph for stock
                    if (size > 0) {
                        g.drawLine(p.x+d.width+2,p.y+10,GRIDLEFT,
                                   scale(updates[0].listenerCount));
                        if (updates[size - 1] != null) {
                            g.drawString(String.valueOf(updates[size - 1].listenerCount),
                                         GRIDLEFT+size*INCR,
                                         scale(updates[size-1].listenerCount));
                        }
                    }
                }

                // draw graph of updates for this stock
```

```
                int x = GRIDLEFT;
                for (int i = 0; i < size; i++) {
                    if (updates[i] != null) {
                        g.fillOval(x-1, scale(updates[i].listenerCount)-1, 4, 4);
                        if ((i < size - 1) && updates[i + 1] != null) {
                            g.drawLine(x, scale(updates[i].listenerCount),
                                       x + INCR, scale(updates[i + 1].listenerCount));
                        }
                    }
                    x += INCR;
                }
            }

    }

    /*
    * Used to scale y-values.
    */
    int scale(float y) {
        return HEIGHT - (int) (y*3+.5) - 50;
        //return HEIGHT - (int) (y*2+.5) - 50;
    }

    /*
    * Make sure that mouseHere is set properly.
    */
    void setMouseHere(boolean display) {
        Enumeration enum = channelTable.elements();
        while (enum.hasMoreElements()) {
            ChannelData data = (ChannelData) enum.nextElement();
            data.cb.mouseHere = display;
        }
    }
}

/**
 * ChannelData contains stock updates and display information.
 */
class ChannelData {

    // channel
    public String id;
    private static int channelCount = 0;

    // update history
    public Vector updates;
    private static int updateCount;
    final static int MAX_UPDATES = 34;

    // display
    public boolean displayed;
    public SmartCheckbox cb;
    public Color color;
    private Color[] colorTable = {Color.black, Color.gray, Color.cyan,
                                  Color.pink, Color.magenta, Color.orange,
                                  Color.blue.brighter(), Color.green,
                                  Color.red.brighter(), Color.gray};

    /**
     * Constructor.
     */
    public ChannelData(String id) {
        this.id = id;
        this.color = colorTable[channelCount++ % colorTable.length];
```

ChannelMonitorApplet.java Thu Jun 17 14:24:02 1999

```
        this.updates = new Vector(MAX_UPDATES);
        displayed = true;
    }

    /*
    * Updates channel status.
    */
    void update(ChannelStatistics channel) {
        synchronized (updates) {
            if (updates.size() == MAX_UPDATES) {
                updates.removeElementAt(0);
            }
            if (updates.size() < updateCount - 1) {
                for (int i = updates.size(); i < updateCount - 1; i++) {
                    updates.addElement(channel);
                }
            }
            updates.addElement(channel);
            updateCount = updates.size();
        }
    }

    /**
     * Resets counters.
     */
    public static void reset() {
        updateCount = 0;
        channelCount = 0;
    }
}

/*
 * A smart checkbox that records whether the mouse is over the checkbox.
 */
class SmartCheckbox extends Canvas {
    ChannelData data;
    boolean state = true;
    ChannelMonitorApplet panel;
    boolean mouseHere = false;

    public boolean haveMouse() {
        return mouseHere;
    }

    /**
     * Constructor.
     */
    public SmartCheckbox(ChannelData data, ChannelMonitorApplet p) {
        this.data = data;
        panel = p;
    }

    public boolean mouseEnter(Event evt, int x, int y) {
        if (state) {
            //panel.setMouseHere(false);
            mouseHere = true;
            panel.repaint();
        }
        return false;
    }

    public boolean mouseExit(Event evt, int x, int y) {
        if (state) {
            mouseHere = false;
```

```
            panel.repaint();
        }
        return false;
    }

    public boolean mouseDown(Event evt, int x, int y) {
        if (state)
            state = false;
        else
            state = true;
        mouseHere = state;
        data.displayed = state;
        repaint();
        panel.repaint();
        return true;
    }

    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.drawLine(4,4,14,4);
        g.drawLine(4,4,4,14);
        g.setColor(Color.gray);
        g.drawLine(5,14,14,14);
        g.drawLine(14,5,14,14);
        g.setColor(data.color);
        g.fillRect(5,5,8,8);
        g.setColor(data.color);
        g.drawString("Ch-" + data.id, 17, 15);
        g.setColor(Color.white);
        if (state) {
            g.fillRect(7, 7, 4, 4);
        }
    }

    public void setState(boolean s) {
        state = s;
        repaint();
    }
}
```

ChannelStatistics.java Thu Jun 17 14:24:26 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Channel Accounting Data]
 *
 * $<marconi.ras.>ChannelStatistics.java -v2.0(prototype version), 1999/05/22 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

/**
 * This class encapsulates the channel accounting information, such as the number
 * of current listeners. It can be extended to include other kinds of data for auditing
 * or periodic logging.
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.Channel
 * @since   prototype v1.0
 */
public class ChannelStatistics implements java.io.Serializable {

    /**
     * Channel Id.
     */
    public String chan_id = "";

    /**
     * Usage. The current number of listeners for the specified channel.
     */
    public int listenerCount = 0;

    /**
     * Constructor.
     */
    public ChannelStatistics(String id, int count) {
        chan_id = id;
        listenerCount = count;
    }
}
```

CommercialSchedule.java Thu Jun 17 14:24:39 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Commercial Schedule Class]
 *
 * $<marconi.ras.>CommercialSchedule.java -v1.0(prototype version), 1998/09/06 $
 * @jdk1.1.7, -riK.
 */
package marconi.ras;

import java.io.*;
import java.util.*;

/**
 * The <code>CommercialSchedule</code> class represents a template for schedule of
 * commercial time slots. Currently, the number of commercial breaks each day and
 * the number of advertisement slots for each break are statically set, but it
 * should be extended to be more dynamic. In any event, it provides several APIs
 * for retrieving proper information from the advertisement database.
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @since   prototype v1.0
 */
public class CommercialSchedule {

    /*
    * An array of commercial breaks.
    */
    private CommercialBreak[] breaks;

    /**
     * Number of breaks per day.
     */
    public int N_BREAKS = 8;

    /*
    * Break counter.
    */
    private int breakCounter = -1;

    /*
    * Time-slot counter
    */
    private int slotCounter = -1;

    /**
     * Creates new instance of commercial schedule. Use default number of breaks.
     */
    public CommercialSchedule() {
        breaks = new CommercialBreak[N_BREAKS];

        // allocate breaks with slots
        for (int i = 0; i < N_BREAKS; i++) {
            breaks[i] = new CommercialBreak();
        }
    }

    /**
     * Creates new instance of commercial schedule. This cannot be used in this
     * version because the advertisement registration protocol assumes the default
     * setting anyways.
     *
     * @param   freq    number of breaks per day.
     */
```

CommercialSchedule.jc Thu Jun 17 14:24:39 19

```
    public CommercialSchedule(int freq) {
        N_BREAKS = freq;
        breaks = new CommercialBreak[N_BREAKS];

        // allocate breaks with slots
        for (int i = 0; i < N_BREAKS; i++) {
            breaks[i] = new CommercialBreak();
        }
    }

    /**
     * Goes to next commercial break.
     */
    public void nextBreak() {
        breakCounter = (breakCounter + 1) % N_BREAKS;
        slotCounter = -1;

    }

    /**
     * Gets the next advertisement id in schedule.
     */
    public String nextSlot() {
        if (breakCounter < 0) {
            nextBreak();
        }
        slotCounter = (slotCounter + 1) % breaks[breakCounter].N_SLOTS;

        return breaks[breakCounter].getCommercial(slotCounter);
    }

    /**
     * Returns the available time slots for advertisement. Each vector element is
     * associated with a commercial break, which has 8 (default) possible slots.
     * These 8 slots are represented by a <code>binary string</code> where a '1'
     * means that the slot is occupied.
     *
     * @return a vector of bytes which represents the available commercial slots.
     */
    public synchronized Vector getTimeSlots() {
        Vector breakList = new Vector();

        for (int i = 0; i < N_BREAKS; i++) {
            breakList.addElement(breaks[i].getBitmap());
        }
        return breakList;
    }

    /**
     * Sets the requested time slot for the given advertisement.
     */
    public boolean setTimeSlot(int break_id, int slot_id, String ad_id) {
        return breaks[break_id].reserveSlot(slot_id, ad_id);
    }
}

/**
 * Each commercial break consists of the following:
 * <ul>
 * <li>the number of slots for this break (does not neccessarily have to
 *     be 8 but the prototype should use this default value);
 * <li>array of slots that contains the commercial ids;
 * </ul>
 * <p>
```

CommercialSchedule.java Thu Jun 17 14:24:39 1999

```
 *
 * @author  ErikK.
 * @version $Revision: 1.0 $
 * @since   prototype v1.0
 */
class CommercialBreak {

    /**
     * The number of commercial slots per each break.
     */
    public final int N_SLOTS = 8;

    /*
    * This prototype version does not use the below parameter...
    *
    private Date breakTime;    // beginning of the commercial break time
    */

    /*
    * 8-bits are used to represent 8 commercials. Each bit maps to a 30 second
    * commercial.
    */
    private int bitmap;

    /*
    * The array of slots that contain advertisement id.
    */
    private String[] slots;

    /*
    * Constructor for commercial break.
    */
    publ.  CommercialBreak() {
        this.bitmap = 0;
        slots = new String[N_SLOTS];
    }

    /*
    * Check to see if all the slots are full.
    */
    protected synchronized boolean isFull() {
        return (bitmap >= (1 << N_SLOTS) - 1) ? true : false;
    }

    /*
    * Get the bitmap representation of the slots.
    */
    protected String getBitmap() {
        return Integer.toBinaryString(bitmap);
    }

    /*
    * Set the bit for the requested time slot. Slots are of course 0-based (0 - 7).
    */
    protected synchronized boolean reserveSlot(int slot, String ad_id) {
        if (isFull()) {
            return false;
        }

        int slot_bit = 1 << slot;
        if ((bitmap & slot_bit) > 0) {
            return false;
        }
        else {
```

CommercialSchedule.ja[redacted] Thu Jun 17 14:24:39 199

```
            bitmap |= slot_bit;
            slots[slot] = ad_id;
            return true;
        }
    }

    /*
    * Reserve any available time slot;
    */
    protected synchronized boolean reserveSlot(String ad_id) {
        if (this.isFull()) {
            return false;
        }

        boolean success = false;
        for (int i = 0; i < N_SLOTS; i++) {
            if (slots[i] == null) {
                bitmap |= 1 << i;
                slots[i] = ad_id;
                success = true;
                break;
            }
        }
        return success;
    }

    /*
    * Get the specific commercial.
    */
    protected synchronized String getCommercial(int slot) {
        return slots[slot];
    }
}
```

RASActionHandler.java [redacted] Jun 17 14:24:54 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RAS Action Handler]
 *
 * S<marconi.ras.>RASActionHandler.java -v2.0(prototype version), 1998/02/17 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.awt.*;
import java.awt.event.*;

/**
 * This class handles two action commends, particularly <code>Ok</code> and
 * <code>Toggle</code> buttons.
 */
public class RASActionHandler {
    public static ActionListener buttonControl = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            if (command.equals("Ok")) {
                RASControls.processCommand();
            }
            else if (command.equals("Toggle")) {
                RASControls.toggleButton();
            }
            else if (command.equals("Add Commercial")) {
                ADControls.processAdd();
            }
            else if (command.equals("Submit Commercials")) {
                ADControls.processSubmit();
            }
            else if (command.equals("Remove Commercials")) {
                ADControls.processRemove();
            }
        }
    };

    public static ActionListener listControl = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (RASControls.actionDisplay.equals("Add")) {
                RASControls.entry_1.setText
                    (RASDirectory.directoryList_1.getItem
                    (RASDirectory.directoryList_1.getSelectedIndex()));
            }
        }
    };
}
```

RASControls.java Thu Jun 17 14:25:05 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RAS GUI Controls]
 *
 * $<marconi.ras.>RASControls.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * This panel contains user interfaces to the applet.
 */
public class RASControls extends Panel {
    public static RASMgrApplet owner;
    public static Choice ids;
    public static TextField entry_1;
    public static Button button_1;
    public static Button button_2;
    public static String actionDisplay;
    private static int WIDTH  = 600;
    private static int HEIGHT = 50;

    /**
     * Instantiates the control panel.
     */
    public RASControls(RASMgrApplet main) {
        owner = main;
        actionDisplay = "Add";

        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.NONE;
        cons.weightx = 0.0;

        // define toggle button
        button_1 = new Button("   " + actionDisplay + "   ");
        button_1.setActionCommand("Toggle");
        button_1.addActionListener(RASActionHandler.buttonControl);
        button_1.setBackground(Color.lightGray);
        //        button_1.setBackground(Color.black);
        button_1.setForeground(Color.black);
        //        button_1.setForeground(Color.red);
        add(button_1);
        validate();

        // channel id
        ids = new Choice();
        updateIDs();
        grid.setConstraints(ids, cons);
        ids.setForeground(Color.black);
        //        ids.setForeground(Color.yellow.darker());
        ids.setBackground(Color.lightGray);
        //        ids.setBackground(Color.black);
        add(ids);
        validate();

        // selected station (multicast address + name)
        entry_1 = new TextField(30);
        grid.setConstraints(entry_1, cons);
        entry_1.setForeground(Color.black);
```

RASControls.java u Jun 17 14:25:05 1999

```
        //          entry_1.setForeground(Color.yellow.da
        entry_1.setBackground(Color.gray.brighter());
        //          entry_1.setBackground(Color.blue.darker().darker());
        add(entry_1);
        validate();

        // add global to local
        cons.gridwidth = GridBagConstraints.REMAINDER;
        button_2 = new Button("  Ok  ");
        button_2.setActionCommand("Ok");
        button_2.addActionListener(RASActionHandler.buttonControl);
        button_2.setBackground(Color.lightGray);
        //          button_2.setBackground(Color.black);
        button_2.setForeground(Color.black);
        //          button_2.setForeground(Color.red);
        grid.setConstraints(button_2, cons);
        add(button_2);
        validate();

        // resize
        setSize(WIDTH, HEIGHT);
    }

    /**
     * Carries out the specified action.
     */
    public static void processCommand() {

        // if action is "Add"
        if (actionDisplay.equals("Add")) {
            String textfield = entry_1.getText();
            String id_str = ids.getSelectedItem();

            if (textfield.indexOf(", ") > 0 && id_str != null) {
                StringTokenizer dir = new StringTokenizer(textfield, ", ");
                String ma = dir.nextToken();
                String name = dir.nextToken();
                int id = Integer.parseInt(id_str);
                if (owner.addChannel(id, ma)) {
                    owner.showMessage("Added new channel...", false);
                    updateIDs();
                }
                else {
                    owner.showMessage("Channel could not be added...", true);
                }
            }
            else {
                owner.showMessage("Select a station from global directory.", true);
            }
        }

        // if action is "Remove"
        else if (actionDisplay.equals("Remove")) {
            String id_str = ids.getSelectedItem();

            if (id_str != null) {
                int id = Integer.parseInt(id_str);
                if (owner.removeChannel(id)) {
                    owner.showMessage("Removed channel " + id + "...", false);
                    updateIDs();
                }
                else {
                    owner.showMessage("Channel could not be removed...", true);
                }
```

```
            }
            else {
                owner.showMessage("No channel to remove...", true);
            }
        }

        // clear input fields
        entry_1.setText("");
    }

    /**
     * Implements the toggle mechanism for the control button;
     */
    public static void toggleButton() {

        // if button label says "Add", change it to "Remove"
        if (actionDisplay.equals("Add")) {
            actionDisplay = "Remove";
            owner.showMessage("Please select a channel to remove...", false);
            updateIDs();
        }

        // if button label says "Remove", change it to "Add"
        else if (actionDisplay.equals("Remove")) {
            actionDisplay = "Add";
            owner.showMessage("Please select a station to add...", false);
            updateIDs();
        }

        button_1.setLabel(actionDisplay);
    }

    /**
     * Updates available channel ids.
     */
    public static void updateIDs() {
        ids.removeAll();

        if (actionDisplay.equals("Remove")) {
            for (int i = 0; i < owner.channelIDs.length; i++) {
                if (owner.channelIDs[i]) {
                    ids.add(String.valueOf(i));
                }
            }
        }
        else {
            for (int i = 0; i < owner.channelIDs.length; i++) {
                if (!owner.channelIDs[i]) {
                    ids.add(String.valueOf(i));
                }
            }
        }
    }
}
```

RASDirectory.java Th Jun 17 14:25:15 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RAS Directory Controls]
 *
 * $<marconi.ras.>RASDirectory.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, -rik.
 */
package marconi.ras;

import java.awt.*;
import java.awt.event.*;
import java.util.Enumeration;
import java.io.*;
import marconi.util.*;

/**
 * This panel contains user interfaces to the applet.
 */
public class RASDirectory extends Panel
    implements Runnable {

    /**
     * This thread periodically downloads global channel announcements (CAP) cache.
     */
    private Thread g_directoryThread = null;

    /**
     * This thread updates the announcements for the locally supported channels.
     */
    private Thread l_directoryThread = null;

    /**
     * The global announcement update interval.
     */
    private static long G_UPDATE = 20000;

    /**
     * The local announcement update interval.
     */
    private static long L_UPDATE = 20000;

    public static RASMgrApplet owner;
    public static Label label_1;
    public static List directoryList_1;
    public static Label label_2;
    public static List directoryList_2;
    private static int WIDTH  = 600;
    private static int HEIGHT = 200;

    /**
     * The file dump.
     */
    public final static File file = new File("_mapsta");

    /**
     * Instantiates the directory display panel.
     */
    public RASDirectory(RASMgrApplet main) {
        owner = main;
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.NONE;
        setFont(new Font("Helvetica", Font.BOLD, 24));
```

RASDirectory.java : Thu Jun 17 14:25:15 1999

```
        // label 1
        cons.weightx = 1.0;
        label_1 = new Label();
        label_1.setText("   Global Channel Directory   ");
        grid.setConstraints(label_1, cons);
        label_1.setForeground(Color.black);
        //        label_1.setForeground(Color.white);
        add(label_1);
        validate();

        // label 2
        cons.gridwidth = GridBagConstraints.REMAINDER;
        label_2 = new Label();
        label_2.setText("   Local Channel Directory   ");
        grid.setConstraints(label_2, cons);
        label_2.setForeground(Color.black);
        //        label_2.setForeground(Color.white);
        add(label_2);
        validate();
        setFont(new Font("Helvetica", Font.PLAIN, 14));

        // list 1 - global
        cons.gridwidth = GridBagConstraints.RELATIVE;
        directoryList_1 = new List(10, false);
        directoryList_1.addActionListener(RASActionHandler.listControl);
        grid.setConstraints(directoryList_1, cons);
        directoryList_1.setForeground(Color.black);
        directoryList_1.setBackground(Color.white);
        //        directoryList_1.setForeground(Color.yellow.brighter());
        //        directoryList_1.setBackground(Color.darkGray);
        add(directoryList_1);
        validate();

        // list 2 - local
        cons.gridwidth = GridBagConstraints.REMAINDER;
        directoryList_2 = new List(10, false);
        //        directoryList_2.addActionListener(RASActionHandler.listControl);
        grid.setConstraints(directoryList_2, cons);
        directoryList_2.setForeground(Color.black);
        directoryList_2.setBackground(Color.white);
        //        directoryList_2.setForeground(Color.yellow.brighter());
        //        directoryList_2.setBackground(Color.darkGray);
        add(directoryList_2);
        validate();

        // resize
        setSize(WIDTH, HEIGHT);

        start();
    }

    /**
     * Starts the directory update.
     */
    public void start() {
        g_directoryThread = new Thread(this);
        g_directoryThread.start();
        l_directoryThread = new Thread(this);
        l_directoryThread.start();
    }

    /**
     * The run methods for the two threads.
     */
```

```
    public void run() {

        // global directory
        while (Thread.currentThread() == g_directoryThread) {
            PrintWriter fout = null;

            try {
                Thread.sleep(G_UPDATE);
                fout = new PrintWriter(new BufferedWriter(new FileWriter(file)));
                Enumeration enum = owner.rasServer.downloadCAP(false).elements();
                Vector2 ip_addr = new Vector2();
                Vector2 name = new Vector2();

                while (enum.hasMoreElements()) {
                    CDPPacket cdp = (CDPPacket) enum.nextElement();
                    ip_addr.addElement(cdp.MADDR);
                    name.addElement(cdp.name);
                    fout.println(cdp.MADDR + " " + cdp.name);
                }

                displayDirectory(ip_addr.toStringArray(), name.toStringArray());
                fout.close();
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }

        // local directory
        while (Thread.currentThread() == l_directoryThread) {
            try {
                Thread.sleep(L_UPDATE);
                Enumeration enum = owner.rasServer.downloadCAP(true).elements();
                Vector2 ip_addr = new Vector2();
                Vector2 name = new Vector2();
                Vector2 id = new Vector2();

                while (enum.hasMoreElements()) {
                    CDPPacket cdp = (CDPPacket) enum.nextElement();
                    ip_addr.addElement(cdp.MADDR);
                    name.addElement(cdp.name);
                    id.addElement(cdp.id);
                }
                displayDirectory(ip_addr.toStringArray(), name.toStringArray(),
                                 id.toStringArray());
            }
            catch (Exception e) {
            }
        }
    }

    protected static void displayDirectory(String[] ip_addr, String[] name) {

        if (directoryList_1.getItemCount() > 0)
            directoryList_1.removeAll();
        for (int i = 0; i < ip_addr.length; i++) {
            directoryList_1.add(ip_addr[i] + ", " + name[i]);
        }
    }

    protected static void displayDirectory(String[] ip_addr, String[] name, String[] id) {
        if (directoryList_2.getItemCount() > 0)
            directoryList_2.removeAll();
        for (int i = 0; i < owner.MAX_CHANNELS; i++) {
```

RASDirectory.java Thu Jun 17 14:25:15 1999

```
            directoryList_2.add(String.valueOf(i));
        }
        for (int i = 0; i < ip_addr.length; i++) {
            directoryList_2.replaceItem(id[i] + " " + ip_addr[i] + ", " + name[i],
                                Integer.parseInt(id[i]));
        }
    }

    protected static void displayDirectory(String ip_addr, String name) {
        directoryList_1.add(ip_addr + ", " + name);
    }

    protected static void displayDirectory(String ip_addr, String name, String id) {
        directoryList_2.replaceItem(id + ip_addr + ", " + name, Integer.parseInt(id));
    }
}
```

RASManager.java Thu[illegible] 17 14:25:23 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RAS Manager Interface]
 *
 * $<marconi.ras.>RASManager.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.rmi.*;

/**
 * This interface is provided for the RAS to write alarming texts at the remote manager.
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.MarconiServer
 * @since   prototype v1.0
 */
public interface RASManager extends Remote {

        /**
         * Write various messages (error messages).
         */
  public voic showMessage(String msg, boolean blink)
       throws RemoteException;
}
```

```
RASMessageBoard.java          Jul 17 14:25:45 1999

/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : (RAS Messageboard Display)
 *
 * $<marconi.ras.>RASMessageBoard.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.awt.*;
import java.net.*;
import java.applet.*;

/**
 * This message borad is used to display messages in a scrolling fashion.
 * The content of the message is updated by directly accessing and changing
 * the public variables <code>text</code> and <code>blink</code>.
 * If <code>blink</code> is set to <code>true</code>, the message stored in
 * <code>text</code> will scroll and also !blink!; this feature is
 * particularly for alarm (error) messages.
 */
public class RASMessageBoard extends Applet
    implements Runnable {

    // publically accessable vars
    public String text = "";
    public boolean blink = false;

    // awt vars
    int shiftCnt = 0 ;
    private Font font;
    private static Color fgcolor = Color.blue;
    private static Color bgcolor = Color.lightGray;
    private Color color = fgcolor;
    private Image offScrImage = null;
    private Graphics offScrGraphics = null;
    private Dimension offScrSize = null;

    Thread thread = null;

    /**
     * Even though this object extends from an applet, it is not to be
     * the main applet and thus is instantiated via a constructor; you may
     * set it to display some initial text;
     */
    public RASMessageBoard(String initial_text) {

        // initialize text field
        this.text = initial_text;

        // measure screen width
        shiftCnt = getSize().width;

        // setup font
        font = new Font("Helvetica", Font.PLAIN, 16);
        setFont(font);
        start();
    }

    /**
     * Start the thread;
     */
    public void start() {
        if(thread == null) {
            thread = new Thread(this);
```

RASMessageBoard.java Thu Jun 17 14:25:45 1999

```
            thread.start();
        }
    }

    /**
     * Stop the thread;
     */
    public void stop() {
        thread = null;
    }

    /**
     * Thread body; blinking feature is implemented by toggling text color
     * b/w background and foreground here after each pause;
     */
    public void run() {
        int i = 0;
        while (Thread.currentThread() == thread) {

            // pause
            try {
                Thread.currentThread().sleep(200);
            }
            catch (InterruptedException e) {
            }

            // blink -> toggle colors b/w black and yellow
            // it remains black for 2 pause periods and yellow for 4 periods
            // otherwise -> stay yellow
            if (blink)
                color = (i++ < 2) ? bgcolor : fgcolor;
            else
                color = fgcolor;

            // update drawing
            repaint();
            i = (i > 6) ? 0 : i;
        }
    }

    /**
     * Update display (move the message to the left).
     */
    public synchronized void update(Graphics g) {
        FontMetrics fMetric;
        fMetric  = getFontMetrics(font);
        Dimension d = getSize();

        // create off-screen image
        if((offScrImage == null) || (d.width != offScrSize.width) ||
           (d.height != offScrSize.height)) {
            offScrImage = createImage(d.width, d.height);
            offScrSize = d;
            offScrGraphics = offScrImage.getGraphics();
            offScrGraphics.setFont(font);
        }

        // setup off-screen image
        offScrGraphics.setColor(bgcolor);
        offScrGraphics.fillRect(0, 0, d.width, d.height);
        offScrGraphics.setColor(color);
        offScrGraphics.setFont(font);
        if ((shiftCnt + fMetric.stringWidth(text)) <= 0)
            shiftCnt = d.width;
```

RASMessageBoard.java Thu Jul 17 14:25:45 1999

```
        // shift left & write message
        shiftCnt = shiftCnt - 5;
        offScrGraphics.drawString(text, shiftCnt, (int) (d.height * 0.65));
        g.drawImage(offScrImage, 0, 0, null);
    }
}
```

RASMgrApplet.java Thu Jun 17 14:26:00 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RAS Manager Applet]
 *
 * $<marconi.ras.>RASMgrApplet.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.applet.*;
import java.awt.*;
import java.util.*;
import java.net.URL;
import java.rmi.*;
import java.rmi.server.*;

/**
 * This applet is used by a RAS manager (operator) who monitors and configures
 * the MarconiServer.
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.MarconiServer
 * @since   prototype v1.0
 */
public class RASMgrApplet extends Applet
    implements RASManager, java.io.Serializable {

    // miscellaneious variables
    private static int width = 0;
    private static int height = 0;
    protected RAS rasServer = null;
    protected int MAX_CHANNELS = 0;
    protected static boolean[] channelIDs;
    final static String obj_name = "marconi.ras.RASMgrApplet";

    // tools
    RASDirectory directory = null;
    RASControls controls = null;
    ADControls ad_controls = null;
    RASMessageBoard msgBoard = null;

    /**
     * Initialize the applet and setup display area.
     */
    public void init() {
        try {
            width = Integer.parseInt(getParameter("APPLWIDTH"));
            height = Integer.parseInt(getParameter("APPLHEIGHT"));

            // lookup RAS (MarconiServer)
            URL URLbase = getDocumentBase();
            System.out.println(obj_name + ".init: locating RAS");
            rasServer = (RAS) Naming.lookup("//" + URLbase.getHost() + ":"
                                            + getParameter("RASPort")
                                            + "/marconi.ras.MarconiServer");

            // init variables
            MAX_CHANNELS = MarconiServer.MAX_CHANNELS;
        }
        catch (Exception e) {
            // fatal error
            System.err.println(obj_name + ".init: ");
            e.printStackTrace();
        }
```

RASMgrApplet.java Tue Jun 17 14:26:00 1999

```
        channelIDs = new boolean[MAX_CHANNELS];
        for (int i = 0; i < channelIDs.length; i++) {
            channelIDs[i] = false;
        }

        // draw display area
        setupDisplay();
    }

    /**
     * Display the applet.
     */
    public void setupDisplay() {
        Label label_1;
        setBackground(Color.lightGray);

        directory = new RASDirectory(this);
        controls = new RASControls(this);
        ad_controls = new ADControls(this);
        msgBoard = new RASMessageBoard("Initializing RAS...");
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();

        // setup grid
        int rowHeights[] = {10, 200, 50, 200, 90};
        grid.rowHeights = rowHeights;
        setLayout(grid);
        cons.fill = GridBagConstraints.BOTH;

        // label 1
        Font font = new Font("Arial", Font.BOLD, 24);
        setFont(font);
        cons.gridwidth = GridBagConstraints.REMAINDER;
        label_1 = new Label("- RAS MANAGER -", Label.CENTER);
        grid.setConstraints(label_1, cons);
        label_1.setForeground(Color.black);
        add(label_1);
        validate();
        setFont(new Font("Arial", Font.PLAIN, 14));

        // add directory lists
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
        grid.setConstraints(directory, cons);
        add(directory);
        validate();

        // add controls
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
        grid.setConstraints(controls, cons);
        add(controls);
        validate();

        // add ad-insertion controls
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
        grid.setConstraints(ad_controls, cons);
        add(ad_controls);
        validate();
```

```
        // add message scroller board
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.weighty = 1.0;
        cons.gridheight = 1;
        grid.setConstraints(msgBoard, cons);
        add(msgBoard);
        validate();

        resize(width, height);
    }

    /**
     * Close down the server when closing applet.
     */
    public void destroy() {

        if (rasServer != null) {
            try {
                rasServer.shutdown();
                remove(directory);
                remove(msgBoard);
                remove(controls);
            }
            catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Register channel with the server.
     */
    public boolean addChannel(int id, String ip) {
        boolean status = false;
        try {
            status = rasServer.registerChannel(id, ip);
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
        if (status) {
            channelIDs[id] = true;
            update_ctrls();
        }
        return status;
    }

    /**
     * Remove channel from the server.
     */
    public boolean removeChannel(int id) {
        boolean status = false;
        try {
            status = rasServer.removeChannel(id);
        }
        catch (RemoteException e) {
        }
        if (status) {
            channelIDs[id] = false;
            update_ctrls();
        }
        return status;
```

RASMgrApplet.java Thu Jun 17 14:26:00 1999

```
    }

    /**
     * Update tools upon action taken.
     */
    private void update_ctrls() {
        //      controls.updateIDs();
        ad_controls.updateIDs();
    }

    /**
     * Write various messages (error messages).
     */
    public void showMessage(String msg, boolean blink) {
        msgBoard.text = msg;
        msgBoard.blink = blink;
    }

    /**
     * Return applet information.
     */
    public String getAppletInfo() {
        return "RAS configuration/management tool";
    }
}
```

RTSPServerControl.java Thu Jun 17 14:26:19 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RTSP Server Controller]
 *
 * $<marconi.ras.>RTSPServerControl.java -v2.0(prototype version), 1998/1/12 $
 * @jdk1.2, ~riK.
 */
package marconi.ras;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * This class provides interfaces to manipulate the remote RTSPServer.
 * <p>
 * @author  ~riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.MarconiServer
 * @since   prototype 1.0
 */
public class RTSPServerControl {

    /*
    * RTSPServer private info.
    */
    private final static String obj_name = "marconi.ras.RTSPServerControl";
    private final static int TOR_OFFSET = 0;
    private final static int M_OFFSET = 4;
    private final static int LM_OFFSET = 8;
    private final static int PATH_OFFSET = 12;


    /**
     * The RTSPServer Host.
     */
    protected String hostname;

    /**
     * The RTSPServer port.
     */
    protected int port;

    /*
    * TCP-Client resources for communicating with the RTSPServer.
    */
    private Socket rtsp_sock = null;
    private BufferedInputStream rtsp_in = null;
    private BufferedOutputStream rtsp_out = null;

    /**
     * Type of request field value <code>start</code>.
     */
    protected final static byte TOR_START = 1;

    /**
     * Type of request field value <code>stop</code>.
     */
    protected final static byte TOR_STOP = 2;

    /**
     * Type of request field value <code>commercial</code>.
     */
    protected final static byte TOR_COMMERCIAL = 3;
```

RTSPServerControl.jav Thu Jun 17 14:26:19 1999

```
    /**
     * Returned status value <code>ok</code>.
     */
    protected final static byte STATUS_OK = 0;

    /**
     * Returned status value <code>error</code>.
     */
    protected final static byte STATUS_ERR = 1;

    /**
     * Returned status value <code>duplicate</code>.
     */
    protected final static byte STATUS_DUP = 2;

    /**
     * Returned status value <code>states full</code>.
     */
    protected final static byte STATUS_FUL = 3;

    /**
     * The number of bytes returned by RTSPServer.
     */
    protected final static int RECEIPT_LEN = 1;

    /**
     * Creates an object that will interface with the RTSPServer.
     */
    public RTSPServerControl(String rtsp_h, int rtsp_p) {
        this.hostname = rtsp_h;
        this.port = rtsp_p;
        try {
            this.rtsp_sock = new Socket(rtsp_h, rtsp_p);
            this.rtsp_in = new BufferedInputStream(rtsp_sock.getInputStream());
            this.rtsp_out = new BufferedOutputStream(rtsp_sock.getOutputStream());
        }
        catch (IOException e) {
        }
    }

    /**
     * Signals the RTSPServer to start a new thread to support the specified channel.
     *
     * @param   g_maddr         global multicast address.
     * @param   l_maddr         local multicast address.
     * @return  byte representing the status returned from RTSPServer.
     */
    public byte startChannel(InetAddress global_addr, InetAddress local_addr) {
        byte[] requestPkt;
        byte[] receiptPkt;
        int n = 0;

        // compose outgoing packet & send
        requestPkt = encode(TOR_START,
                            global_addr.getAddress(),
                            local_addr.getAddress(),
                            null);
        try {
            rtsp_out.write(requestPkt, 0, requestPkt.length);
        }
        catch (IOException e) {
            System.err.println(obj_name + ".startChannel: " + e.getMessage());
            e.printStackTrace();
        }
```

```
        // read the status returned by RTSPServer
        receiptPkt = new byte[RECEIPT_LEN];
        try {
            n = rtsp_in.read(receiptPkt, 0, RECEIPT_LEN);
        }
        catch (IOException e) {
            System.err.println(obj_name + ".startChannel: " + e.getMessage());
            e.printStackTrace();
        }
        if (n < RECEIPT_LEN) {
            System.err.println(obj_name + ".startChannel: returned byte is corrupted.");
            return (STATUS_ERR);
        }

        // return status
        return (receiptPkt[0]);
    }

    /**
     * Signals the RTSPServer to stop the specified channel.
     *
     * @param   l_maddr          local multicast address.
     * @return  byte representing the status returned from RTSPServer.
     */
    public byte stopChannel(InetAddress local_addr) {
        byte[] requestPkt;
        byte[] receiptPkt;
        int n = 0;

        // compose outgoing packet & send
        requestPkt = encode(TOR_STOP, null, local_addr.getAddress(), null);
        try {
            rtsp_out.write(requestPkt, 0, requestPkt.length);
        }
        catch (IOException e) {
            System.err.println(obj_name + ".stopChannel: " + e.getMessage());
            e.printStackTrace();
        }

        // read the status returned by RTSPServer
        receiptPkt = new byte[RECEIPT_LEN];
        try {
            n = rtsp_in.read(receiptPkt, 0, RECEIPT_LEN);
        }
        catch (IOException e) {
            System.err.println(obj_name + ".stopChannel: " + e.getMessage());
            e.printStackTrace();
        }
        if (n < RECEIPT_LEN) {
            System.err.println(obj_name + ".stopChannel: returned byte is corrupted.");
            return (STATUS_ERR);
        }

        // return status
        return (receiptPkt[0]);
    }

    /**
     * Signals the RTSPServer to play the given commercial.
     *
     * @param   l_maddr          local multicast address.
     * @param   path                    commercial file path.
     * @return  byte representing the status returned from RTSPServer.
```

RTSPServerControl.jav[redacted] Thu Jun 17 14:26:19 1999

```
     */
    public byte playCommercial(InetAddress local_addr, String path) {
        byte[] requestPkt;
        byte[] receiptPkt;
        int n = 0;

        // compose outgoing packet & send
        requestPkt = encode(TOR_COMMERCIAL, null, local_addr.getAddress(), path);
        try {
            rtsp_out.write(requestPkt, 0, requestPkt.length);
        }
        catch (IOException e) {
            System.err.println(obj_name + ".playCommercial: " + e.getMessage());
            e.printStackTrace();
        }

        // read the status returned by RTSPServer
        receiptPkt = new byte[RECEIPT_LEN];
        try {
            n = rtsp_in.read(receiptPkt, 0, RECEIPT_LEN);
        }
        catch (IOException e) {
            System.err.println(obj_name
                               + ".playCommercial: " + e.getMessage());
            e.printStackTrace();
        }
        if (n < RECEIPT_LEN) {
            System.err.println(obj_name
                               + ".playCommerical: returned byte is corrupted.");
            return (STATUS_ERR);
        }

        // return status
        return (receiptPkt[0]);
    }

    /**
     * Encodes the input parameters into a TCP packet.
     *
     * @param   tor                 type of request.
     * @param   m_addr  global multicast address for a station.
     * @param   lm_addr local multicast address for a station.
     * @param   path            commercial file.
     * @return  the encoded byte array buffer.
     */
    static byte[] encode(int tor, byte[] m_addr, byte[] lm_addr, String path) {
        int path_len = (tor == TOR_COMMERCIAL) ? path.length() : 0;
        byte[] buffer = new byte[PATH_OFFSET +
                                 ((tor == TOR_COMMERCIAL) ? path_len : 1)];

        // type of request
        buffer[TOR_OFFSET + 0] = (byte) ((tor << 16) >>> 24);
        buffer[TOR_OFFSET + 1] = (byte) ((tor << 24) >>> 24);

        // path length
        buffer[TOR_OFFSET + 2] = (byte) ((path_len << 16) >>> 24);
        buffer[TOR_OFFSET + 3] = (byte) ((path_len << 24) >>> 24);

        // M: global multicast address
        if (tor == 1) {
            System.arraycopy(m_addr, 0, buffer, M_OFFSET, m_addr.length);
        }

        // LM: local multicast address
```

RTSPServerControl.java Thu Jun 17 14:26:19 1999

```
        System.arraycopy(lm_addr, 0, buffer, LM_OFFSET, lm_a

        // commercial file path...
        if (tor == 3) {
            byte[] pathbuf = path.getBytes();

            for (int i = 0; i < pathbuf.length; i++) {
                buffer[12 + i] = pathbuf[i];
            }
        }

        return buffer;
    }

    /**
     * Decode the status value.
     */
    public static String decodeStatus(byte status) {
        switch (status) {
        case 0: return "OK";
        case 1: return "ERROR";
        case 2: return "DUPLICATE";
        case 3: return "STATES_FULL";
        default: return "UNKNOWN_STATUS=" + status;
        }
    }
}
```

LAS.java Thu Jun 17 [illegible]:26:3[illegible] 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Internet Radio Server (DIRS) : [LAS interface]
 *
 * $<marconi.>LAS.java -v1.0(prototype version), 98/8/19.
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.rmi.*;
import java.util.Vector;

/**
 * The LAS (local advertisement server) remote interface provides interfaces
 * for the advertising company, to store and broadcast local commercials.
 */
public interface LAS extends Remote {

    //    public int uploadCommercial(byte[] file)
    //        throws RemoteException;

    //    public Vector reviewTimeSlots(int channel)
    //        throws RemoteException;

    /**
    * Tell the ad server to reserve a slot.
    */
    //    public boolean buyCommercialTime(int channel, String ad_id)
    //        throws RemoteException;
}
```

MarconiServer.java Thu Jun 17 14:26:53 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RAS Implementation]
 *
 * $<marconi.ras.>MarconiServer.java -v3.0(prototype version), 1998/12/22 $
 * @jdk1.2, -riK.
 */
package marconi.ras;

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
import java.net.*;
import java.util.*;
import java.io.*;
import marconi.util.*;

/**
 * The <code>MarconiServer</code> class implements the interfaces <code>RAS</code>
 * (radio antenna server) and <code>LAS</code> (local advertisement server).
 * As a RAS, it should manage a particular set of multicast channels that are
 * active and allow the IRCs (internet radio clients) to be able to tune to
 * each channel. The LAS server provides an API-like interface to the advertising
 * companies. It also maintains a local database to store the commercials.
 * <p>
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.Channel
 * @since   prototype 1.0
 */
public class MarconiServer extends UnicastRemoteObject
    implements RAS, Runnable { // this version does not implement LAS yet...

    /*
     * Radio Antenna Server local id
     */
    final String name = "marconiNet: RAS, the radio antenna server/MarconiServer";
    final static String obj_name = "marconi.ras.MarconiServer";
    private String hostname = null;

    /**
     * The hard-coded port number for local RMI registry.
     */
    public final static int RMI_PORT = 5678;

    /**
     * The multicast address used for global announcement of CDP packets.
     */
    public final static String GLOBAL_CAP = "225.3.0.0";

    /**
     * The multicast address used for local announcement of CDP packets.
     */
    public final static String LOCAL_CAP = "225.3.0.1";

    /**
     * The port used for channel announcement protocol (CAP).
     */
    public final static int CAP_PORT = 7777;

    /**
     * The port used for multicast communication between RSC and RASs.
     */
    public final static int RSC_PORT = 8910;
```

MarconiServer.java Thu Jun 17 14:26:53 1999

```
    /**
     * The port used for multicast communication between RAS and IRCs.
     */
    public final static int IRC_PORT = 8910;

    /**
     * The default TCP port used for communicating with the RTSPServer.
     */
    public final static int RTSP_PORT = 8765;

    /**
     * The ttl used for multicast from RSC to RASs.
     */
    public final static int GLOBAL_TTL = 128;

    /**
     * The ttl used for multicast from RAS to IRCs.
     */
    public final static int LOCAL_TTL = 16;

    /**
     * The maximum number of channels that can be supported locally (finite number
     * of channels).
     */
    public final static int MAX_CHANNELS = 20;

    /**
     * The maximum media content payload length in bytes (=RTP payload length).
     */
    public final static int MAX_PAYLOADLEN = 4096;

    /**
     * The maximum RTP header length.
     */
    public final static int MAX_RTPHDRLEN = 20;

    /**
     * This thread receives channel announcements and maintains the channel
     * directory database (analogous to session directory -Sdr).
     */
    private volatile Thread directoryThread = null;

    /**
     * This thread announces the channel descriptions to the local listeners.
     */
    private volatile Thread announcerThread = null;

    /**
     * This thread is started along with the <code>directoryThread</code>. It is
     * used to generate the advertisement schedule.
     */
    private volatile Thread advertiseThread = null;

    /**
     * RAS channel registry / database.
     */
    private Channel[] channelRegistry = null;

    /**
     * Station multicast address (CDP) to channel ID mapping.
     */
    private Hashtable channelMapper = null;
    private final static Integer NOT_SUPPORTED = new Integer(-1);
```

```
    /**
     * Channel Announcement Protocol Cache.
     */
    private Hashtable capCache = null;

    /**
     * The RTSPServer remote controller.
     */
    private RTSPServerControl rtspServer = null;

    /**
     * The local multicast address dispenser (one instance per RAS).
     */
    private MaddrDispenser l_maddrRegistry = null;

    /**
     * The local multicast address for local station's content.
     */
    private InetAddress l_maddr_local = null;

    /**
     * The local station's channel id.
     */
    public final static String LCCALSTA = "LOCAL";

    /**
     * The files containing local station's program announcement.
     */
    private final static String LOCALSTA_CDP = "_localcdp";
    private final static String LOCALSTA_SCHED = "_localsched";

    /**
     * Time interval between each Marconi process (30 seconds).
     */
    private final static long INTERVAL = 10000; // reduced for demo

    /**
     * Channel Announcement Protocol (CAP) resources.
     */
    private MulticastSocket cap_receiver = null;
    private MulticastSocket cap_sender = null;
    private boolean SOCKET1_IN_USE = false;
    private boolean SOCKET2_IN_USE = false;

    /**
     * The RSA public key.
     */
    private byte[] publicKey = {(byte) 0x0};

    /**
     * The RSA private key.
     */
    private byte[] privateKey = {(byte) 0x0};

    /**
     * Instantiate the MarconiServer of RAS (radio antenna server) with its
     * default settings. It also establishes a connection to the RTSPServer.
     */
    public MarconiServer(String rtsp_h, int rtsp_p) throws RemoteException {
        try {
            // initialzie
            hostname = InetAddress.getLocalHost().getHostName();
            rtspServer = new RTSPServerControl(rtsp_h, rtsp_p);
            channelRegistry = new Channel[MAX_CHANNELS];
```

MarconiServer.java Thu Jun 17 14:26:53 1999

```
            channelMapper = new Hashtable();
            capCache = new Hashtable();
            cap_sender = new MulticastSocket();
            cap_receiver = new MulticastSocket(CAP_PORT);
            l_maddrRegistry = new MaddrDispenser();
            l_maddr_local = l_maddrRegistry.next();

            // join CAP multicast group
            cap_receiver.joinGroup(InetAddress.getByName(GLOBAL_CAP));
        }
        catch (Exception e) {
        }
        start();
    }

    /**
     * Registers a station into a channel slot. These slots are indexed through
     * the channel id's.
     *
     * @param   c   the channel id.
     * @param   ma  global multicast address used for station broadcast.
     * @return  true if the station/channel is successfully registered and false
     *          otherwise.
     */
    public synchronized boolean registerChannel(int c, String ma)
        throws RemoteException {
        // if channel is not already taken and its updated cache exists
        if (channelRegistry[c] == null && capCache.containsKey(ma)) {

            // add new channel to database
            channelMapper.put(ma, new Integer(c));
            try {
                InetAddress l_maddr = l_maddrRegistry.next();
                channelRegistry[c] = new Channel(c, l_maddr);
            }
            catch (MaddrException e) {
                return false;
            }

            // initialize channel from the cache
            channelRegistry[c].read_cdp((CDPPacket) capCache.get(ma));
            channelRegistry[c].init(publicKey, privateKey);

            // remove new channel from the global cache
            //capCache.remove(ma);

            System.out.println
                (obj_name + ".registerChannel: created channel-" + c);

            return true;
        }

        // if channel in use---
        else {
            System.err.println(obj_name +
                               ".register.Channel: cannot create channel-" + c);
            return false;
        }
    }

    /**
     * Removes a channel from the database.
     *
     * @param   c   the channel id to be removed.
```

```
     * @return  true if the station/channel is successfully
     *                  otherwise.
     */
    public synchronized boolean removeChannel(int c)
        throws RemoteException {
        // if the channel exists
        if (channelRegistry[c] != null) {

            // stop the channel thread and free resource
            if (channelRegistry[c].isOnline()) {
                channelRegistry[c].destroy();
            }
            l_maddrRegistry.remove(channelRegistry[c].l_maddr);

            // remove channel from database
            channelMapper.put(channelRegistry[c].g_maddr.getHostAddress(),
                              NOT_SUPPORTED);
            channelRegistry[c] = null;
            System.out.println(obj_name + ".removeChannel: removed channel-" + c);
            return true;
        }

        // if the channel doesn't exist---
        else {
            System.err.println(obj_name +
                               ".remove.Channel: cannot remove channel-" + c);
            return false;
        }
    }

    /*
     * @deprecated  replace by RTCP signaling.
     *
     * This class does not actually implement the audio playing mechanism. It
     * Return the status of this request. This method is merely used as means
     * of finding out who's listening to what. The next version should replace
     * this module with a more scalable approach such as by utilizing the
     * RTCP signals. Currently, this RMI request is also being used to trigger
     * the actual broadcasting. If the requested channel is broadcasting
     * already, nothing else is done. If not, the MarconiServer initiates the
     * broadcasting procedures (i.e. start listening to the global multicast
     * address and redirecting the stream locally).
     *
     * @param   c   the channel id.
     * @return  true if successful, false otherwise.
     *
    public synchronized boolean playChannel(int c)
        throws RemoteException {

        // if channel exists
        if (channelRegistry[c] != null) {

            // if channel not already started, signal RTSPServer to start it
            channelRegistry[c].HIT_COUNT++;
            if (!channelRegistry[c].isOnline()) {
                try {
                    channelRegistry[c].start();
                    System.out.println(obj_name + ".playChannel: channel-" + c
                                       + " started.");
                }
                // catch TooManyChannelThreadsException
                //   & IllegalThreadStateException
                catch (Exception e) {
                    System.err.println(obj_name +
```

```
                                        ".playChannel:
                    return false;
                }
            }

            // return the local multicast address of the channel
            return true;
        }
        else {
            System.err.println(obj_name +
                               ".playChannel: cannot play channel-" + c);
            return false;
        }
    }
    */

    /**
     * Stops all broadcasting channels and terminates this RAS.
     */
    public void shutdown() throws RemoteException {
        synchronized (channelRegistry) {
            for (int i = 0; i < MAX_CHANNELS; i++) {
                if (channelRegistry[i] != null) {
                    removeChannel(i);
                }
            }
        }
        stop();
        System.exit(0);
    }

    /**
     * Starts the MarconiServer.
     */
    public void start() {
        directoryThread = new Thread(this);
        directoryThread.start();
        announcerThread = new Thread(this);
        announcerThread.start();
        advertiseThread = new Thread(this);
        advertiseThread.start();
    }

    /**
     * Stops the MarconiServer.
     */
    public void stop() {
        directoryThread = null;
        announcerThread = null;
        advertiseThread = null;

        // release sockets and leave announcement group
        try {
            while (SOCKET1_IN_USE) {
                Thread.sleep(3);
            }
            cap_receiver.leaveGroup(InetAddress.getByName(GLOBAL_CAP));
            cap_receiver.close();
            cap_sender.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    /**
     * Returns the channel usage statistics.
     *
     * @return  an array of channel statistics. The array size is the
     *          <code>MAX_CHANNELS</code>. There maybe <code>null</code> entries in the
     *          array for the channel slots that are not supported.
     */
    public ChannelStatistics[] getStatistics() {
        ChannelStatistics[] stats = new ChannelStatistics[MAX_CHANNELS];
        synchronized (channelRegistry) {
            for (int i = 0; i < MAX_CHANNELS; i++) {
                if (channelRegistry[i] != null) {
                    stats[i] = channelRegistry[i].audit();
                }
            }
        }
        return stats;
    }

    /**
     * Adds commercial list.
     *
     * @param   ad_list an array of <code>String</code>s that refers to the
     *                  commercial filenames.
     * @return  whether the request was successful.
     */
    public boolean submitCommercialList(String[] ad_list) {
        try {
            for (int i = 0; i < ad_list.length; i++) {
                StringTokenizer st = new StringTokenizer(ad_list[i]);
                int id = Integer.parseInt(st.nextToken());
                channelRegistry[id].addCommercial(st.nextToken());
            }
            for (int i = 0; i < MAX_CHANNELS; i++) {
                if (channelRegistry[i] != null) {
                    channelRegistry[i].genCommercialFile();
                }
            }
            return true;
        }
        catch (Exception e) {
            System.err.println(obj_name +
                               ".submitCommercialList: illegal list format");
            return false;
        }
    }

    /*
     * This <code>run</code> method starts the MarconiServer.
     */
    public void run() {

        // cache update hour
        long hour = System.currentTimeMillis();

        /*
        * Global directory thread running.
        */
        while (Thread.currentThread() == directoryThread) {

            /*
            * Receive CDP packets and maintain channel database.
            */
```

MarconiServer.java Thu Jun 17 14:26:53 1999

```
            SOCKET1_IN_USE = true;
            try {
                DatagramPacket recv_pkt = CDPPacket.compose();
                cap_receiver.receive(recv_pkt);
                CDPPacket cdp = new CDPPacket(recv_pkt);

                synchronized (channelRegistry) {

                    // create new mapping
                    if (!channelMapper.containsKey(cdp.MADDR)) {
                        channelMapper.put(cdp.MADDR, NOT_SUPPORTED);
                    }

                    // update announcement appropriately
                    Integer Id = (Integer) channelMapper.get(cdp.MADDR);
                    if (Id.equals(NOT_SUPPORTED)) {
                        capCache.put(cdp.MADDR, cdp);
System.out.println(obj_name + ".run: new cdp cached for " + cdp.MADDR);
                    }
                    else {
                        channelRegistry[Id.intValue()].read_cdp(cdp);
System.out.println(obj_name + ".run: local channel's cdp cache refreshed");
                    }
                }
            }
            catch (Exception e) {
                e.printStackTrace();
            }
            SOCKET1_IN_USE = false;

            // time to refresh cache (daily)
            long current_hour = System.currentTimeMillis();
            if (current_hour > hour + Timestamp.DAY) {
System.out.println(obj_name + ".run: routine -refresh local cache");
                refresh_cache();
                hour += Timestamp.DAY;
            }

            /*
            * Interval b/w each loop for receiving global channel directory.
            */
            if (!Thread.interrupted()) {
                try {
System.out.println(obj_name + ".run: ----------  -  ----------");
                    Thread.sleep(INTERVAL);
                }
                catch (InterruptedException e) {
                }
            }
        }

        /*
        * Local Directory thread running.
        */
        while (Thread.currentThread() == announcerThread) {

            /*
            * For each installed local channel send out announcements.
            */
            for (int i = 0; i < MAX_CHANNELS && announcerThread != null; i++) {
                if (channelRegistry[i] == null) {
                    continue;
                }
```

```
            CDPPacket local_cdp = channelRegistry[i].writ
            DatagramPacket send_pkt = local_cdp.compose(LOCAL_CAP, CAP_PORT);
            try {
                cap_sender.send(send_pkt, (byte) LOCAL_TTL);
            }
            catch (IOException e) {
                System.out.println(obj_name + ".run: ");
                e.printStackTrace();
            }

            // sleep between every announcement (do not flood network)
            try {
                Thread.sleep(5000);
            }
            catch (InterruptedException e) {
            }
        }

        /*
        * Announce local track programming.
        */
        if (announcerThread != null) {
        try {
            CDPPacket localsta_cdp = write_cdp();
            DatagramPacket send_pkt = localsta_cdp.compose(LOCAL_CAP, CAP_PORT);
            cap_sender.send(send_pkt, (byte) LOCAL_TTL);
        }
        catch (Exception e) {
            System.out.println(obj_name + ".run: ");
            e.printStackTrace();
        }
        }

        /*
        * Interval b/w each loop for sending local channel directory.
        */
        if (!Thread.interrupted()) {
            try {
                Thread.sleep(INTERVAL);
            }
            catch (InterruptedException e) {
            }
        }
    }

    /*
    * Advertise thread running.
    */
    while (Thread.currentThread() == advertiseThread) {

        /*
        * Dump each channel's commercial queue (list of commercials to play)
        * to a file so that it can be used by the LAIP (local advertisement
        * insertion protocol).
        */
        for (int i = 0; i < MAX_CHANNELS; i++) {
            if (channelRegistry[i] != null) {
                channelRegistry[i].genCommercialFile();
            }
        }

        /*
        * Interval b/w each loop for generating commercial files.
        */
```

MarconiServer.java Thu Jun 17 14:26:53 1999

```
            if (!Thread.interrupted()) {
                try {
                    Thread.sleep(Timestamp.DAY / 4);
                }
                catch (InterruptedException e) {
                }
            }
            //Thread.yield();
        }
    }

    /**
     * Removes old cache entries.
     */
    private void refresh_cache() {
        long current_hour = System.currentTimeMillis();

        synchronized (capCache) {
            Enumeration capList = capCache.elements();
            while (capList.hasMoreElements()) {
                CDPPacket cdp = (CDPPacket) capList.nextElement();
                if (current_hour > cdp.timeStamp + CDPPacket.TTL) {
                    capCache.remove(cdp.MADDR);
                }
            }
        }
    }

    /**
     * Provides local and global cache of channel announcements for
     * immediate download (instead of waiting for the periodic announcement).
     *
     * @param   local   <code>true</code> if requesting for a local announcement download.
     * @return  list of channel descriptions (CDPPackets) in <code>Vector</code>.
     * @see     marconi.util.CDPPacket
     */
    public Vector downloadCAP(boolean local) throws RemoteException {

        // return local announcements
        if (local) {
            synchronized (channelRegistry) {
                Vector cdp_list = new Vector();

                for (int i = 0; i < MAX_CHANNELS; i++) {
                    if (channelRegistry[i] != null) {
                        CDPPacket cdp = channelRegistry[i].write_cdp();
                        cdp_list.addElement(cdp);
                    }
                }
                return cdp_list;
            }
        }
        // return global announcements
        else {
            synchronized (capCache) {
                Vector vec = new Vector();
                Enumeration enum = capCache.elements();
                while (enum.hasMoreElements()) {
                    vec.addElement(enum.nextElement());
                }
                return vec;
            }
        }
    }
```

```
    /**
     * Creates a CDP announcement for the local station track.
     *
     * @return  a channel description of the local track.
     * @see     marconi.util.CDPPacket
     */
    private CDPPacket write_cdp() throws AnnouncementException {
        CDPPacket cdp = null;
        BufferedReader fin = null;
        String schedule = "";

        try {
            cdp = new CDPPacket(LOCALSTA_CDP);
            fin = new BufferedReader(new FileReader(LOCALSTA_SCHED));
        }
        catch (Exception e) {
            throw new AnnouncementException
                (obj_name + ".write_cdp: the local schedule file cannot be opened.");
        }
        while (fin != null) {
            String line = null;
            try {
                if ((line = fin.readLine()) == null) {
                    fin.close();
                    break;
                }
            }
            catch (IOException e) {
                break;
            }
            if (line.length() > 0) {
                String program = "";
                StringTokenizer st = new StringTokenizer(line);
                Calendar cal = Calendar.getInstance();

                try {
                    for (int i = 0; i < 2; i++) {
                        cal.set(Calendar.DAY_OF_WEEK, Integer.parseInt(st.nextToken()));
                        cal.set(Calendar.HOUR_OF_DAY, Integer.parseInt(st.nextToken()));
                        cal.set(Calendar.MINUTE, Integer.parseInt(st.nextToken()));
                        cal.set(Calendar.SECOND, Integer.parseInt(st.nextToken()));
                        program += String.valueOf(cal.getTime().getTime()) + "|";
                    }
                    program += st.nextToken() + "|-";
                    schedule += (schedule.length() > 0) ? "," : "" + program;
                }
                catch (Exception e) {
                    throw new AnnouncementException
                        (obj_name + ".write_cdp: invalid local schedule file.");
                }
            }
        }
        cdp.id = LOCALSTA;
        cdp.date = Timestamp.get_midnight();
        cdp.MADDR = this.l_maddr_local.getHostAddress();
        cdp.MPORT = this.IRC_PORT;
        cdp.MTTL = this.LOCAL_TTL;
        cdp.schedule = schedule;

        return cdp;
    }

    /*
```

MarconiServer.java Thu Jun 17 14:26:53 1999

```
    * The main method executes the server setup proce...
    */
   public static void main(String args[]) throws RemoteException {

       // check arguments (rtsp server)
       if (args.length != 2) {
           System.err.println("usage: \n"
                              + "MarconiServer <RTSP hostname> <RTSP port>");
           System.exit(-1);
       }
       System.setErr(System.out);

       // install a security manager
       System.setSecurityManager(new RMISecurityManager());

       // setup and start the server on local host
       try {
           LocateRegistry.createRegistry(RMI_PORT);
           MarconiServer marconiServer = new MarconiServer(args[0],
                                                           Integer.parseInt(args[1]));
           Naming.rebind("//:" + RMI_PORT + "/" + obj_name, marconiServer);
           System.out.println(marconiServer.hostname
                              + " bound in registry at port " + RMI_PORT);
       }
       catch (Exception e) {
           System.err.println(obj_name + ".main: " + e.getMessage());
           e.printStackTrace();
       }
   }
}
```

IRCControls.java Thu Jan 17 14:28:20 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [IRC GUI Controls]
 *
 * $<marconi.irc.>IRCControls.java -v2.0(prototype version), 1999/02/15 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.io.*;

/**
 * This panel contains user interfaces to the applet.
 */
public class IRCControls extends Panel {
    public static TextField entry_1;
    private static int WIDTH  = 600;
    private static int HEIGHT = 100;

    /**
     * Instantiates the control panel.
     */
    public IRCControls() {
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.NONE;
        cons.weightx = 0.0;

        // selected station (id + name)
        entry_1 = new TextField(40);
        grid.setConstraints(entry_1, cons);
        entry_1.setForeground(Color.yellow.darker());
        entry_1.setBackground(Color.blue.darker().darker());
        add(entry_1);
        validate();

        // resize
        setSize(WIDTH, HEIGHT);
    }
}
```

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [IRC Directory Controls]
 *
 * $<marconi.ras.>IRCDirectory.java -v2.0(prototype version), 1999/04/20 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.awt.*;
import java.awt.event.*;
import java.util.Hashtable;
import java.net.*;
import java.io.*;
import marconi.ras.MarconiServer;
import marconi.util.CDPPacket;

/**
 * This panel contains user interfaces to the applet.
 */
public class IRCDirectory extends Panel
    implements Runnable {

    /**
     * This thread updates the announcements for the locally supported channels.
     */
    private Thread updateThread = null;
    private static long L_UPDATE = 7500;

    public static IRCUsrApplet owner;
    public static Label label_1;
    public static List directoryList_1;
    private static int WIDTH  = 600;
    private static int HEIGHT = 400;

    /**
     * Instantiates the directory display panel.
     */
    public IRCDirectory(IRCUsrApplet main) {
        owner = main;
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();
        setLayout(grid);
        cons.fill = GridBagConstraints.NONE;
        cons.weightx = 1.0;

        // label 1
        cons.weightx = 1.0;
        cons.gridwidth = GridBagConstraints.REMAINDER;
        label_1 = new Label();
        label_1.setText("Local Channel Directory");
        grid.setConstraints(label_1, cons);
        label_1.setForeground(Color.white);
        add(label_1);
        validate();

        // list 1 - global
        cons.gridwidth = GridBagConstraints.REMAINDER;
        directoryList_1 = new List(20, false);
        directoryList_1.addActionListener(IRCActionHandler.listControl);
        grid.setConstraints(directoryList_1, cons);
        directoryList_1.setForeground(Color.yellow.brighter());
        directoryList_1.setBackground(Color.darkGray);
        add(directoryList_1);
        validate();
```

IRCDirectory.java Tue Jun 17 14:28:27 1999

```
        // resize
        setSize(WIDTH, HEIGHT);
        start();
    }

    /**
     * Starts the directory update.
     */
    public void start() {
        updateThread = new Thread(this);
        updateThread.start();
    }

    /**
     * The run methods for the two threads.
     */
    public void run() {

        // local directory
        while (Thread.currentThread() == updateThread) {
            try {
                Thread.sleep(L_UPDATE);
                Hashtable cdp_lookup = owner.getCache();

                if (directoryList_1.getItemCount() > 0) {
                    directoryList_1.removeAll();
                }
                for (int i = 0; i < owner.MAX_CHANNELS; i++) {
                    String Id = String.valueOf(i);
                    if (cdp_lookup.containsKey(Id)) {
                        CDPPacket cdp = (CDPPacket) cdp_lookup.get(Id);
                        directoryList_1.add(cdp.id + " " + cdp.name,
                                            Integer.parseInt(cdp.id));
                    }
                    else {
                        directoryList_1.add(Id);
                    }
                }
                if (cdp_lookup.containsKey(MarconiServer.LOCALSTA)) {
                    CDPPacket cdp = (CDPPacket) cdp_lookup.get(MarconiServer.LOCALSTA);
                    directoryList_1.add(cdp.name);
                }
            }
            catch (Exception e) {
                System.err.println("marconi.irc.IRCDirectory.run:");
                e.printStackTrace();
            }
        }
    }
}
```

IRCUsrApplet.java Thu Jun 17 14:28:50 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [IRC User Applet]
 *
 * $<marconi.irc.>IRCUsrApplet.java -v2.0(prototype version), 1999/04/20 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.applet.*;
import java.awt.*;
import java.util.*;
import java.net.*;
//import java.rmi.*;
//import java.rmi.server.*;
import marconi.util.*;
import marconi.util.rtsp.IRC;
import marconi.ras.RAS;
import marconi.ras.MarconiServer;

/**
 * This applet is used by an IRC user.
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.ras.MarconiServer
 * @since   prototype v1.0
 */
public class IRCUsrApplet extends Applet
    implements java.io.Serializable, Runnable {

    /**
     * Session Announcement Protocol (SAP) resources.
     * Interfaced via. Channel Directory/Description Protocol (CDP).
     */
    private MulticastSocket cap_receiver = null;
    protected Hashtable capCache = null;

    /**
     * This thread updates the announcements for the locally supported channels.
     */
    private Thread directoryThread = null;
    private static long L_UPDATE = 10000;

    // miscellaneious variables
    private static int width = 0;
    private static int height = 0;
    //protected RAS rasServer = null;
    protected int MAX_CHANNELS = 20;
    final static String obj_name = "marconi.ras.IRCUsrApplet";

    // tools
    IRCDirectory directory = null;
    IRCControls controls = null;
    IRC listener = null;

    /**
     * Initialize the applet and setup display area.
     */
    public void init() {
        try {
            width = Integer.parseInt(getParameter("APPLWIDTH"));
            height = Integer.parseInt(getParameter("APPLHEIGHT"));

            // lookup RAS (MarconiServer)
```

IRCUserApplet.java Tue Jun 17 14:28:50 1999

```
            //URL URLbase = getDocumentBase();
            //System.out.println(obj_name + ".init: locating server");
            //rasServer = (RAS) Naming.lookup("//" + getParameter("RASHost")
            //                                + ":" + getParameter("RASPort")
            //                                + "/marconi.ras.MarconiServer");

            // init variables
            //MAX_CHANNELS = rasServer.getMaxChannels();
            capCache = new Hashtable();
        }
        catch (Exception e) {
            // fatal error
            System.err.println(obj_name + ".init: ");
            e.printStackTrace();
        }

        // join CAP multicast group
        try {
            cap_receiver = new MulticastSocket(MarconiServer.CAP_PORT);
            cap_receiver.joinGroup(InetAddress.getByName(MarconiServer.LOCAL_CAP));
        }
        catch (Exception e) {
            System.out.println(obj_name + ".init:");
            e.printStackTrace();
        }

        // draw display area
        setupDisplay();

        // start
        listener = new IRC();
        directoryThread = new Thread(this);
        directoryThread.start();
    }

    /**
     * Display the applet.
     */
    public void setupDisplay() {
        setBackground(Color.black);

        directory = new IRCDirectory(this);
        controls = new IRCControls();
        GridBagLayout grid = new GridBagLayout();
        GridBagConstraints cons = new GridBagConstraints();

        // setup grid
        int rowHeights[] = {400, 100};
        grid.rowHeights = rowHeights;
        setLayout(grid);
        cons.fill = GridBagConstraints.BOTH;

        // add directory lists
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
        grid.setConstraints(directory, cons);
        add(directory);
        validate();

        // add controls
        cons.gridwidth = GridBagConstraints.REMAINDER;
        cons.weightx = 1.0;
        cons.gridheight = 1;
```

```
        grid.setConstraints(controls, cons);
        add(controls);
        validate();

        resize(width, height);
    }

    /**
     * Free resources when closing applet.
     */
    public void destroy() {
        // release sockets and leave announcement group
        try {
            cap_receiver.leaveGroup(InetAddress.getByName(MarconiServer.LOCAL_CAP));
            cap_receiver.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        stopChannel();
        directoryThread = null;

        //try {
        //    rasServer.terminate();
        //}
        //catch (RemoteException e) {
        //    e.printStackTrace();
        //}

        remove(directory);
        remove(controls);
    }

    /**
     * Run method.
     */
    public void run() {

        // cache update hour
        long hour = System.currentTimeMillis();

        /*
        * Global directory thread running.
        */
        while (Thread.currentThread() == directoryThread) {

            /*
            * Receive CDP packets and maintain channel database.
            */
            try {
                DatagramPacket recv_pkt = CDPPacket.compose();
                cap_receiver.receive(recv_pkt);

                CDPPacket cdp = new CDPPacket(recv_pkt);
                System.out.println(obj_name + ".run: cdp parsed");

                // update announcement appropriately
                if (!capCache.containsKey(cdp.id)) {
                    capCache.put(cdp.id, cdp);
                    System.out.println(obj_name + ".run: new cdp cached for channel-"
                                       + cdp.id);
                }
            }
            catch (Exception e) {
```

IRCUsrApplet.java Tue Jun 17 14:28:50 1999

```
                e.printStackTrace();
            }

            // time to refresh cache (daily)
            long current_hour = System.currentTimeMillis();
            if (current_hour > hour + Timestamp.DAY) {
                System.out.println(obj_name + ".run: routine -refreshing directory cache.");
                refresh_cache();
                hour += Timestamp.DAY;
            }

            /*
            * Interval b/w each loop for receiving local channel directory.
            */
            try {
                Thread.sleep(L_UPDATE);
            }
            catch (InterruptedException e) {
            }
        }
    }

    /**
     * Removes old cache entries.
     */
    protected void refresh_cache() {
        long current_hour = System.currentTimeMillis();

        synchronized (capCache) {
            Enumeration capList = capCache.elements();
            while (capList.hasMoreElements()) {
                CDPPacket cdp = (CDPPacket) capList.nextElement();
                if (current_hour > cdp.timeStamp + CDPPacket.TTL) {
                    capCache.remove(cdp.id);
                }
            }
        }
    }

    /**
     * Inform server that the specified channel is being listen to. This
     * RMI based triggering is very inefficient and not scalable. So
     * alternate approach based on RTCP should replace this.
     */
    public boolean playChannel(int id) {
        boolean status = false;

        if (capCache.containsKey(String.valueOf(id))) {
            CDPPacket cdp = (CDPPacket) capCache.get(String.valueOf(id));
            try {
                // kill previous thread if running
                listener.stop();
                /*
                * The below statement is commented out because the listener
                * is now capable of sending RTCP signals for triggering.
                *
                status = rasServer.playChannel(id);
                */
                status = true; // replace above
                listener.start(cdp.MADDR, cdp.MPORT);
            }
            catch (Exception e) {
                e.printStackTrace();
                return false;
```

```
            }
            return status;
        }
        else {
            return false;
        }
    }

    /**
     * Stops listening to whatever is playing.
     */
    public void stopChannel() {
        listener.stop();
    }

    /**
     * Returns the current state of the local channel announcement cache.
     */
    protected synchronized Hashtable getCache() {
        return capCache;
    }

    /**
     * Return applet information.
     */
    public String getAppletInfo() {
        return "IRC listener tool";
    }
}
```

IRCActionHandler.java Thu Jan 17 14:29:06 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [IRC Action Handler]
 *
 * $<marconi.irc.>IRCActionHandler.java -v2.0(prototype version), 1999/04/21 $
 * @jdk1.2, -riK.
 */
package marconi.irc;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * This class handles actions taken by IRC user.
 */
public class IRCActionHandler {
    public static ActionListener listControl = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            IRCControls.entry_1.setText
                (IRCDirectory.directoryList_1.getItem
                (IRCDirectory.directoryList_1.getSelectedIndex()));
            String textfield = IRCControls.entry_1.getText();
            StringTokenizer dir = new StringTokenizer(textfield, " ");
            int id = Integer.parseInt(dir.nextToken());
            if (IRCDirectory.owner.playChannel(id)) {
                System.out.println("marconi.irc.IRCActionHandler" +
                                   ".actionPerformed: playing channel "
                                   + id + ".");
            }
            else {
                IRCDirectory.owner.stopChannel();
                System.err.println("marconi.irc.IRCActionHandler" +
                                   ".actionPerformed: error listening.");
            }
        }
    };
}
```

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [CDP Announcer]
 *
 * $<marconi.ras.>Announcer.java -v2.0(prototype version), 1999/04/12 $
 * @jdk1.2, -riK.
 */
package marconi.rsc;

import java.net.*;
import java.util.*;
import java.io.*;
import marconi.ras.MarconiServer;
import marconi.util.*;

/**
 * The <code>Announcer</code> makes periodic announcements via. CDP (SAP/SDP).
 * <p>
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.CDPPacket
 * @since   prototype 1.0
 */
public class Announcer implements Runnable {
    final static String obj_name = "marconi.rsc.Announcer";
    String hostname = "";

    /**
     * This thread announces the channel descriptions to the local listeners.
     */
    private volatile Thread announcerThread = null;

    /**
     * Time interval between each Marconi process (30 seconds).
     */
    private final static long INTERVAL = 30000;

    /**
     * Channel Announcement Protocol (CAP) resources.
     */
    private MulticastSocket cap_sender = null;

    private String file = "";

    /**
     * Constructor.
     */
    public Announcer(String file) {
        try {
            hostname = InetAddress.getLocalHost().getHostName();
            cap_sender = new MulticastSocket();
        }
        catch (Exception e) {
        }
        this.file = file;
        start();
    }

    /**
     * Starts the Announcer.
     */
    public void start() {
        announcerThread = new Thread(this);
        announcerThread.start();
    }
```

Announcer.java Jun 17 14:29:45 1999

```
    /**
     * Stops the Announcer.
     */
    public void stop() {
        announcerThread = null;

        // release sockets and leave announcement group
        try {
            cap_sender.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * This <code>run</code> method starts the Announcer.
     */
    public void run() {

        /*
        * Announcer thread running.
        */
        while (Thread.currentThread() == announcerThread) {

            /*
            * Send out each announcement.
            */
            CDPPacket cdp = null;
            try {
                cdp = new CDPPacket(file);
            }
            catch (AnnouncementException e) {
                e.printStackTrace();
            }
            DatagramPacket send_pkt = cdp.compose(MarconiServer.GLOBAL_CAP,
                                                  MarconiServer.CAP_PORT);

            if (cdp.id != null) {
                try {
                    cap_sender.send(send_pkt, (byte) MarconiServer.GLOBAL_TTL);
                }
                catch (IOException e) {
                    System.out.println(obj_name + ".run: ");
                    e.printStackTrace();
                }

                System.out.println(obj_name + ".run: announcement sent to "
                                   + MarconiServer.GLOBAL_CAP + "/"
                                   + MarconiServer.CAP_PORT);
            }

            /*
            * Interval b/w each loop for sending local channel directory.
            */
            try {
                Thread.sleep(INTERVAL);
            }
            catch (InterruptedException e) {
            }
        }
    }
```

```
    /*
    * The main method executes the server setup process.
    */
    public static void main(String args[]) {
        // check arguments (rtsp server)
        if (args.length != 1) {
            System.err.println("Usage: \n"  + "Announcer <CDP file>");
            System.exit(1);
        }

        System.setErr(System.out);

        // Setup and start the server on local host
        Announcer announcer = new Announcer(args[0]);
    }
}
```

RSC.java Thu Jun 17 [redacted]9:52 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [RSC Interface]
 *
 * $<marconi.ras.>RSC.java -v2.0(prototype version), 1999/05/14 $
 * @jdk1.2, -riK.
 */
package marconi.rsc;

import java.rmi.*;
import java.util.Vector;

/**
 * The <code>RSC</code> RMI interface provides security/payment APIs for RAS.
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @since   prototype v1.0
 */
public interface RSC extends Remote {

    /**
     * Accept RAS' public key and include this RAS for SEK distribution.
     * $enroll(byte[] publicKey) $
     */
    public int enroll(byte[] pubkey)
        throws RemoteException;
}
```

CDPPacket.java Thu J 17 :4:30:23 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Channel Descriptions Class]
 *
 * $<marconi.ras.>CDPPacket.java -v4.0(prototype version), 1999/05/05 $
 * @jdk1.2, -riK.
 */
package marconi.util;

import java.io.*;
import java.net.*;
import java.util.StringTokenizer;
import marconi.util.sd.*;

/**
 * This class encapsulates and parses the Channel Description Protocol (CDP)
 * Packet. It acts as an transparent interface to the SAP/SDP protocol (the
 * utilization of protocols SAP/SDP is hidden from the users of CDP).
 * <p>
 * Included in the packet are a subset of the following fields in any order:
 * <ul>
 * <li>name
 * <li>category
 * <li>description
 * <li>origin
 * <li>language
 * <li>id
 * <li>maddr
 * <li>mport
 * <li>mttl
 * <li>seklist
 * <li>date*
 * <li>schedule*
 * </ul>
 * <p>
 * The channel description can be created from a file or any other object that
 * can be converted to a text stream. The format is given below.
 * <p>
 * Each field is to be contained in a line. Each line is to start with the field
 * name followed by '=', and then the field value (i.e. "category=news").
 * <p>
 * The fields marked * are optional if the CDP packet is not being used to announce
 * program schedules. If the CDP packet is used for program schedule announcement,
 * all the fields must be specified. (Note: the current version only allows one-day
 * scheduling per an announcement. Multiple CDP packets should be used to schedule
 * for different dates) There is no version number, nor is there a session id. A new
 * announcement can simply replace the previously received ones. The different
 * announcements are distinguished by examining the multicast address and the date.
 * If one wishes to schedule ahead of time, the date field should be used to indicate
 * to which date the schedule field is assigned.
 * <p>
 * The <code>schedule</code> field should be in the following format:
 * <p><blockquote><pre>
 *         schedule=&lt;program1&gt;,&lt;program2&gt;,&lt;program3&gt;, ...etc.
 * </pre></blockquote>
 * <p>
 * separating each sub-field with a comma (',').
 * <p>
 * The <code>date</code> field is represented by specifying the number of
 * milliseconds since the standard base time known as "the epoch", namely
 * January 1, 1970, 00:00:00 GMT. GMT (Greenwich Mean Time) is the Internet
 * conventional reference time zone and it is synchronized with the UTC
 * (Coordinated Universal Time) milliseconds. The receivers of this packet can
 * convert this to the appropriate local time. The data type long (64-bit integer
 * in Java) stores these milliseconds. Note that SDP uses the Network Time Protocol
```

CDPPacket.java Jun 17 14:30:23 1999

```
 * (NTP) timestamps and SDP essentially converts UTC t- ....
 * <p>
 * Each program is constructed as:
 * <p><blockquote><pre>
 *          &lt;start-time&gt;|&lt;end-time&gt;|&lt;program-title&gt;|&lt;description&gt
 * </pre></blockquote>
 * <p>
 * separating each sub-field with a vertical line ('|'). Again, UTC milliseconds
 * should be used to specify the times.
 * <p>
 * Lastly, the seklist is the concatenated list of the radio station's session
 * encryption key (SEK), each time encrypted with the public key that belongs to
 * different RASs (if global announcement) or IRCs (if local announcement). The
 * encrypted values in binary are PEM-encoded and the delimiter '|' is used to
 * separate them. Currently CDPPacket is implemented to support RC4 stream cipher;
 * it is a symmetric encryption algorithm with variable key size and fast performance
 * in software. Another nice feature about it is that it is fairly exportable from US,
 * which is necessary for global usage of MaroniNet.
 * <p>
 * It is necessary that all fields be included (except for the optional ones), although
 * it is not required to have them in order.
 * <p>
 *
 * @author  ~riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.AnnouncementException
 * @see     marconi.util.Timestamp
 * @since   prototype v1.0
 */                                                                    **
public class CDPPacket implements Serializable {                      **
    final static String obj_name = "marconi.ras.CDPPacket";

    /**
     * The station name.
     */
    public String name = null;

    /**
     * The content category (music|news|sports|...).
     */
    public String category = null;

    /**
     * The content description.
     */
    public String description = null;

    /**
     * The origin of the content (regional ID).
     */
    public String origin = null;

    /**
     * The language of choice.
     */
    public String language = null;

    /**
     * The multicast address.
     */
    public String MADDR = null;

    /**
     * The multicast port.
```

```
     */
    public int MPORT = 8888;

    /**
     * The multicast ttl (set to local scope by default).
     */
    public int MTTL = 16;

    /**
     * The array of PEM-encoded (base64) session encryption keys.
     */
    public String[] SEKLIST = null;

    /**
     * The channel id. Must be the local hostname if used for global
     * announcement.
     */
    public String id = null;

    /**
     * The date being scheduled.
     */
    public long date = -1;

    /**
     * The radio station's content schedule (list of programs)
     */
    public String schedule = null;

    /**
     * The date and time of the packet created.
     */
    public long timeStamp = -1;

    /**
     * The life time of CDP packets (expiration time).
     */
    public final static long TTL = Timestamp.DAY;

    /**
     * The maximum CDP packet buffer length (currently 4K bytes).
     */
    public final static int MAX_BUFLEN = 4096;

    /**
     * Raw data containing the byte-array representation of the packet, if it has
     * one.
     */
    private byte[] data = null;

    /**
     * Creates an empty CDP packet.
     */
    public CDPPacket() {
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Creates CDP packet from a (received) datagram.
     *
     * @param   packet  the datagram packet that contains CDP.
     */
    public CDPPacket(DatagramPacket packet) throws AnnouncementException {
        parse(packet);
```

```
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Creates a CDP packet from a file.
     *
     * @param   file    pathname to where the CDP-file is located.
     */
    public CDPPacket(String file) throws AnnouncementException {
        parse(file);
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Initializes this CDP packet by parsing a datagram (UDP) packet.
     *
     * @param   packet   a datagram packet.
     */
    public void parse(DatagramPacket packet) throws AnnouncementException {

        // get session description
        SAPPacket sap = null;
        SDPPacket sdp = null;
        try {
            sap = new SAPPacket(packet.getData());
            sdp = new SDPPacket(sap.payload);
        }
        catch (MalformedSDException e) {
            throw new AnnouncementException
                (obj_name + ".parse: the received announcement cannot be parsed.");
        }

        this.name = sdp.name;
        this.description = sdp.info;
        for (int i = 0; i < sdp.attribute.length; i++) {
            String attr = sdp.attribute[i];
            if (attr == null) {
                continue;
            }
            if (attr.startsWith("lang:")) {
                this.language = attr.substring(attr.indexOf(':') + 1).trim();
            }
            else if (attr.startsWith("cat:")) {
                this.category = attr.substring(attr.indexOf(':') + 1).trim();
            }
            else if (attr.startsWith("X-orig:")) {
                this.origin = attr.substring(attr.indexOf(':') + 1).trim();
            }
            else if (attr.startsWith("X-seklist:")) {
                Vector2 seklist = new Vector2();
                StringTokenizer st = new StringTokenizer
                    (attr.substring(attr.indexOf(':') + 1).trim(), "|");
                while (st.hasMoreTokens()) {
                    seklist.addElement(st.nextToken().trim());
                }
                this.SEKLIST = seklist.toStringArray();
            }
            else if (attr.startsWith("X-date:")) {
                String date_str = attr.substring(attr.indexOf(':') + 1).trim();
                this.date = Long.parseLong(date_str);
            }
            else if (attr.startsWith("X-sched:")) {
                this.schedule = attr.substring(attr.indexOf(':') + 1).trim();
            }
```

CDPPacket.java Thu J[redacted]17 11:30:23 1999 5

```
        }
        this.MADDR = sdp.connection.address;
        this.MTTL = Integer.parseInt(sdp.connection.ttl);
        this.MPORT = Integer.parseInt(sdp.media[0].getPort());
        this.id = sdp.origin.session_id;
    }

    /**
     * Initializes this CDP packet by parsing from a file.
     *
     * @param   file_path   name and path of the CDP announcement file.
     */
    public void parse(String file_path) throws AnnouncementException {

        /*
        * Open file.
        */
        File file = new File(file_path);
        BufferedReader file_in = null;
        try {
            file_in = new BufferedReader(new FileReader(file));
        }
        catch (FileNotFoundException e) {
            throw new AnnouncementException
                (obj_name + ".parse: the CDP file cannot be opened.");
        }

        /*
        * Read-in the file line by line.                                    **
        */                                                                  **
        while (file_in != null) {

            // parse a line
            String line = null;
            try {
                line = file_in.readLine();
            }
            catch (IOException e) {
                break;
            }
            if (line == null) { // break if eof
                try {
                    file_in.close();
                }
                catch (IOException e) {
                }
                break;
            }
            else if (line.startsWith("name=")) {
                this.name = line.substring(line.indexOf('=') + 1).trim();
            }
            else if (line.startsWith("category=")) {
                this.category = line.substring(line.indexOf('=') + 1).trim();
            }
            else if (line.startsWith("description=")) {
                this.description = line.substring(line.indexOf('=') + 1).trim();
            }
            else if (line.startsWith("origin=")) {
                this.origin = line.substring(line.indexOf('=') + 1).trim();
            }
            else if (line.startsWith("language=")) {
                this.language = line.substring(line.indexOf('=') + 1).trim();
            }
            else if (line.startsWith("maddr=")) {
```

CDPPacket.java Thu Jun 17 14:30:23 1999

```
            this.MADDR = line.substring(line.index
        }
        else if (line.startsWith("mport=")) {
            String port_str = line.substring(line.indexOf('=') + 1).trim();
            this.MPORT = Integer.parseInt(port_str);
        }
        else if (line.startsWith("mttl=")) {
            String ttl_str = line.substring(line.indexOf('=') + 1).trim();
            this.MTTL = Integer.parseInt(ttl_str);
        }
        else if (line.startsWith("id=")) {
            this.id = line.substring(line.indexOf('=') + 1).trim();
        }
        else if (line.startsWith("seklist=")) {
            Vector2 seklist = new Vector2();
            StringTokenizer st = new StringTokenizer
                (line.substring(line.indexOf('=') + 1).trim(), "|");
            while (st.hasMoreTokens()) {
                seklist.addElement(st.nextToken().trim());
            }
            this.SEKLIST = seklist.toStringArray();
        }
        else if (line.startsWith("date=")) {
            String date_str = line.substring(line.indexOf('=') + 1).trim();
            this.date = Long.parseLong(date_str);
        }
        else if (line.startsWith("schedule=")) {
            this.schedule = line.substring(line.indexOf('=') + 1).trim();
        }
        else {
            /*
            * Uncomment below if you want strict formating of the files.
            */
            throw new AnnouncementException
                (obj_name + ".parse: the CDP file cannot be parsed");
        }
    }
}

/**
 * Composes a datagram packet that represents this CDP packet going out to the
 * specified address and port number. This packet is UDP encapsulated SAP/SDP
 * session announcemnet protocol. All appropriate fields, not defined as
 * <code>null</code>, are used to make up this packet.
 *
 * @param   addr    destination address string.
 * @param   port    destination port number.
 * @return  a datagram packet consisting of the valid CDP (SAP/SDP) fields.
 */
public DatagramPacket compose(String addr_str, int port) {
    Vector2 temp_vec = new Vector2();
    SAPPacket sap = new SAPPacket();
    SDPPacket sdp = new SDPPacket();

    /*
    * compose sdp
    */
    // o
    sdp.origin.username = "cdp";
    sdp.origin.session_id = id;
    sdp.origin.version = String.valueOf(Timestamp.get_current());
    sdp.origin.network_type = "IN";
    sdp.origin.address_type = "IP4";
    try {
```

```
            sdp.origin.address = InetAddress.getLocalHost();
        }
        catch (UnknownHostException e) {
        }
        // s
        if (name != null) {
            sdp.name = name;
        }
        // i
        if (description != null) {
            sdp.info = description;
        }
        // c
        if (MADDR != null) {
            sdp.connection = new SDPConnection();
            sdp.connection.network_type = "IN";
            sdp.connection.address_type = "IP4";
            sdp.connection.address = MADDR;
            sdp.connection.ttl = String.valueOf(MTTL);
        }
        // a
        if (category != null) {
            temp_vec.addElement("cat:" + category);
        }
        if (origin != null) {
            temp_vec.addElement("X-orig:" + origin);
        }
        if (language != null) {
            temp_vec.addElement("lang:" + language);
        }
        if (SEKLIST != null && SEKLIST[0] != null) {
            String temp_str = SEKLIST[0];
            for (int i = 1; i < SEKLIST.length; i++) {
                if (SEKLIST[i] != null) {
                    temp_str += "|" + SEKLIST[i];
                }
            }
            temp_vec.addElement("X-seklist:" + temp_str);
        }
        if (date != -1) {
            temp_vec.addElement("X-date:" + String.valueOf(date));
        }
        if (schedule != null) {
            temp_vec.addElement("X-sched:" + schedule);
        }
        sdp.attribute = temp_vec.toStringArray();
        // t -permanent session
        sdp.active = new String[1];
        sdp.active[0] = "0 0";
        // m
        sdp.media[0] = new SDPMedia("audio", String.valueOf(MPORT), "RTP/AVP");

        /*
        * compose sap
        */
        sap.message_type = SAPPacket.MT_ANNOUNCE;
        sap.encryption = false;
        sap.compression = false;
        sap.auth_headerlen = 0;
        sap.msgid_hash = 0;
        try {
            sap.source = InetAddress.getLocalHost().getAddress();
        }
        catch (UnknownHostException e) {
```

CDPPacket.java Jun 17 14:30:23 1999

```
        }
        sap.payload = sdp.compose();
        this.data = sap.compose();

        InetAddress addr = null;
        try {
            addr = InetAddress.getByName(addr_str);
        }
        catch (UnknownHostException e) {
        }

        return new DatagramPacket(data, data.length, addr, port);
    }

    /**
     * Composes an ascii file that represents this CDP packet. All appropriate
     * fields, not defined as <code>null</code>, are used to make up this
     * packet.
     *
     * @param   String      filename (path)
     */
    public void compose(String filename) {
        File file = new File(filename);
        PrintWriter fout = null;

        try {
            fout = new PrintWriter(new BufferedWriter(new FileWriter(file)));
        }
        catch (IOException e) {
        }
        if (name != null) {
            fout.println("name=" + name);
        }
        if (description != null) {
            fout.println("description=" + description);
        }
        if (category != null) {
            fout.println("category=" + category);
        }
        if (origin != null) {
            fout.println("origin=" + origin);
        }
        if (language != null) {
            fout.println("language=" + language);
        }
        if (MADDR != null) {
            fout.println("maddr=" + MADDR);
        }
        if (MPORT != -1) {
            fout.println("mport=" + MPORT);
        }
        if (MTTL != -1) {
            fout.println("mttl=" + MTTL);
        }
        if (id != null) {
            fout.println("id=" + id);
        }
        if (SEKLIST != null && SEKLIST[0] != null) {
            String temp_str = SEKLIST[0];
            for (int i = 1; i < SEKLIST.length; i++) {
                if (SEKLIST[i] != null) {
                    temp_str += "|" + SEKLIST[i];
                }
            }
```

```
            fout.println("seklist=" + temp_str);
        }
        if (date != -1) {
            fout.println("date=" + String.valueOf(date));
        }
        if (schedule != null) {
            fout.println("schedule=" + schedule);
        }
        try {
            fout.close();
        }
        catch (Exception e) {
        }
    }

    /**
     * Composes a datagram packet that represents an empty CDP packet.
     * This packet should be instantiated for receiving purposes. It automatically
     * generates a packet with the internal buffer of the maximum length.
     *
     * @return  a datagram packet with empty buffer.
     */
    public static DatagramPacket compose() {
        byte[] buf = new byte[MAX_BUFLEN];

        return new DatagramPacket(buf, buf.length);
    }
}
```

MaddrDispenser.java Jun 17 14:30:38 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [MaddrDispenser Class]
 *
 * $<marconi.maddr.>MaddrDispenser.java -v1.0(prototype version), 1998/11/11 $
 * @jdk1.1.7, -riK.
 */
package marconi.util;

import java.io.*;
import java.net.*;
import java.util.Hashtable;

/**
 * The <code>MaddrDispenser</code> class distributes a domain-wide multicast
 * address for each station, where domain refers to reachability of RAS. Unlike
 * the class <code>MaddrServer</code> it creates multicast channels between RAS
 * and IRCs. Thus, each RAS must include an instance of this object. And
 * assuming a carefully designed domain edges, different instances can utilize
 * the same addresses.
 * <p>
 * We can use administratively scoped address space here, which will eliminate
 * the problem of RAS-coverage (domain) overlaps. But for simplicity and to avoid
 * having to configure the organizational boundary routers, we simply assume
 * for now that there is a local RAS running and use "site-local" scoping
 * (IPv4 TTL=15).
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.MaddrException
 * @since   prototype v1.0
 */
public class MaddrDispenser {

    /*
     * Here we use a contiguous address block from the TTL scoped
     * multicast address spectrum:
     *     [225.2.0.0 : 225.2.255.255] (site-local iif TTL=15)
     * Note that it avoids the administratively scoped IPv4 multicast space:
     *     [239.0.0.0 : 239.255.255.255]
     */
    final static int MADDR_LOWBOUND   = 0x0000;   // [225.2].0.0
    final static int MADDR_UPPERBOUND = 0xFFFF;   // [225.2].255.255
    final static String MADDR_PREFIX  = "225.2";  // 225.0.0.0/16
    final static String RTCP_PREFIX   = "225.4";  //

    Hashtable registeredMaddrs = null;

    /**
     * Creates a new instance of <code>MaddrDispenser</code> where the next
     * multicast address to be allocated is initialized to the lower bound
     * of the address space. All addresses are recycled.
     */
    public MaddrDispenser() {
        registeredMaddrs = new Hashtable();
    }

    /**
     * Allocates and returns a new multicast address. It will first try to fill-in
     * any holes within its address space. In other words, it recycles freed
     * addresses.
     *
     * @return      the new multicast address.
     * @exception   MaddrException if it runs out of the given multicast address
```

MaddrDispenser.java Thu Jun 17 14:30:38 1999

```
     *              pool.
     */
    public synchronized InetAddress next() throws MaddrException {

        // find an empty multicast address space
        int maddr = MADDR_LOWBOUND;
        InetAddress inet_maddr = null;
        while (registeredMaddrs.contains(new Integer(maddr))
               && ++maddr <= MADDR_UPPERBOUND);

        // return newly assigned address
        if (maddr <= MADDR_UPPERBOUND) {
            try {
                inet_maddr = InetAddress.getByName(toString(maddr));
            }
            catch (UnknownHostException e) {
                e.printStackTrace();
            }
            registeredMaddrs.put(inet_maddr, new Integer(maddr));
            return inet_maddr;
        }
        else {
            throw new MaddrException("Multicast address out of bound.");
        }
    }

    /**
     * Free the address. Return the multicast address back to its pool.
     *
     * @param           inet_maddr      the address to be freed.
     */
    public synchronized void remove(InetAddress inet_maddr) {
        registeredMaddrs.remove(inet_maddr);
    }

    /*
    * Construct the full IP address format.
    */
    protected String toString(int addr) {
        return MADDR_PREFIX + "." + ((addr >>> 8) & 0xFF) + "." + (addr & 0xFF);
    }

    /**
     * Return RTCP multicast address of the given RTP multicast address. The details
     * of mapping RTP to RTCP address is hidden.
     *
     * @param rtp_maddr the RTP multicast address.
     */
    public static String rtcp_map(InetAddress rtp_maddr) {
        byte[] raw = rtp_maddr.getAddress();

        return RTCP_PREFIX + "." + ((raw[2] << 24) >>> 24) + "." + ((raw[3] << 24) >>> 24);
    }

    /**
     * Return IPv4 address as 32bit integer.
     *
     * @param maddr the inet address to be converted.
     */
    public static int addr_to_int(InetAddress maddr) {
        byte[] raw = maddr.getAddress();
        int int32 = raw[0] << 24;
        int32 |= (raw[1] << 24) >>> 8;
        int32 |= (raw[2] << 24) >>> 16;
```

```
        int32 |= (raw[3] << 24) >>> 24;
        return int32;
    }
}
```

MaddrServer.java Th[illegible]un 17 14:30:53 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [MaddrServer Class]
 *
 * $<marconi.util.>MaddrServer.java -v1.0(prototype version), 1998/10/12 $
 * @jdk1.2, -riK.
 */
package marconi.util;

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
import java.io.*;
import java.net.*;
import java.util.Hashtable;

/**
 * The <code>MaddrServer</code> class manages the global multicast address
 * allocation to the Radio Station Client (RSC) channels. The current version
 * does not implement distributed approach. Therefore, <code>MaddrServer</code>
 * acts as a centralized server where the RSCs can obtain their multicast
 * addresses via a specific request (RMI request).
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.MaddrException
 * @since   prototype 1.0
 */
public class MaddrServer extends UnicastRemoteObject
    implements MaddrServerInterf {

    final static String obj_name = "marconi.util.MaddrServer";

    /**
     * The port number for multicast address.
     */
    public final static int MADDRSERV_PORT = 8889;

    /*
    * Here we use a contiguous address block from the TTL scoped
    * multicast address spectrum:
    *     [225.1.0.0 : 225.1.255.255] (global iff 128 < TTL < 255)
    * Note that it avoids the administratively scoped IPv4 multicast space:
    *     [239.0.0.0 : 239.255.255.255]
    */
    final static int MADDR_LOWBOUND = 0x0000;     // [225.1].0.0
    final static int MADDR_UPPERBOUND = 0xFFFF;   // [225.1].255.255
    final static String MADDR_PREFIX  = "225.1";  // 225.0.0.0/16

    Hashtable registeredMaddrs = null;

    /**
     * Initializes the multicast address server.
     */
    public MaddrServer() throws RemoteException {
        registeredMaddrs = new Hashtable();
    }

    /**
     * Allocates and returns a new multicast address. It will first try to fill-in
     * any holes within its address space. In other words, it recycles freed
     * addresses.
     *
     * @return          the new multicast address.
```

MaddrServer.java Thu Jun 17 14:30:53 1999

```
     * @exception       MaddrException if it runs out
     *                  pool.
     */
    public synchronized InetAddress next()
        throws RemoteException, MaddrException {

        // find an empty multicast address space
        int maddr = MADDR_LOWBOUND;
        InetAddress inet_maddr = null;
        while (registeredMaddrs.contains(new Integer(maddr))
               && ++maddr <= MADDR_UPPERBOUND);

        // return newly assigned address
        if (maddr <= MADDR_UPPERBOUND) {
            try {
                inet_maddr = InetAddress.getByName(toString(maddr));
            }
            catch (Exception e) {
                e.printStackTrace();
            }
            registeredMaddrs.put(inet_maddr, new Integer(maddr));
            return inet_maddr;
        }
        else {
            throw new MaddrException("Multicast address out of bound.");
        }
    }

    /**
     * Free the address. Return the multicast address back to its pool.
     *
     * @param       inet_maddr  the address to be freed.
     */
    public synchronized void remove(InetAddress inet_maddr)
        throws RemoteException {
        registeredMaddrs.remove(inet_maddr);
    }

    /*
    * Construct the full IP address format.
    */
    protected String toString(int addr) {
        return MADDR_PREFIX + "." + ((addr >>> 8) & 0xFF) + "." + (addr & 0xFF);
    }

    /**
     * The main method executes the server setup process.
     */
    public static void main(String args[]) throws RemoteException {

        // check arguments (rtsp server)
        if (args.length != 0) {
            System.err.println("usage: \n" + "MaddrServer");
            System.exit(-1);
        }

        System.setErr(System.out);

        // Create and install a security manager
        System.setSecurityManager(new RMISecurityManager());

        // Setup and start the server on local host
        try {
            LocateRegistry.createRegistry(MADDRSERV_PORT);
```

MaddrServer.java Thu Jun 17 14:30:53 1999 3

```
            MaddrServer maddrServer = new MaddrServer();
            Naming.rebind("//:" + MADDRSERV_PORT + "/" + obj_name, maddrServer);
            System.out.println(InetAddress.getLocalHost()
                               + " bound in registry at port "
                               + MADDRSERV_PORT);
        }
        catch (Exception e) {
            System.err.println(obj_name + ".main: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
/*
 * EXTENSION OF...
 *
 * @(#)Vector.java       1.60 98/03/30
 *
 * Copyright 1994-1998 by Sun Microsystems, Inc.,
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Sun Microsystems, Inc. ("Confidential Information").  YOU
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Sun.
 */
package marconi.util;

/**
 * This class extends java.util.Vector so that it's capable of returning its
 * elements as an array of java.lang.String.
 *
 * @autor   -riK.
 * @see     java.util.Vector
 */
public class Vector2 extends java.util.Vector {

    /**
     * Constructs an empty vector so that its internal data array
     * has size <tt>10</tt> and its standard capacity increment is
     * zero.
     */
    public Vector2() {
        super(10);
    }

    /**
     * Returns a string array representation of this Vector, containing
     * the String representation of each element.
     */
    public synchronized String[] toStringArray() {
        String[] result = new String[elementCount];
        System.arraycopy(elementData, 0, result, 0, elementCount);
        return result;
    }
}
```

MaddrServerInterf.java Thu Jun 17 14:31:23 1999

```
/* marconiNet - Internet Radio Network
 * Distributed Internet Radio Server (DIRS) : [Maddr Server interface]
 *
 * $<marconi.util.>MaddrServerInterf.java -v1.0(prototype version), 98/8/19.
 * @jdk1.2, -riK.
 */
package marconi.util;

import java.rmi.*;
import java.net.InetAddress;

/**
 * This interface provides centralized distribution of global multicast addresses.
 */
public interface MaddrServerInterf extends Remote {

  public InetAddress next()
       throws RemoteException, MaddrException;

  public void remove(InetAddress maddr)
       throws RemoteException;
}
```

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : (Base64 Wrapper Class)
 *
 * $<marconi.util.>Base64.java -v1.0(prototype version), 1999/05/13 $
 * @jdk1.2, -riK.
 */
package marconi.util;

import java.util.*;

/**
 * This class is a Base64 wrapper for ASCII representation of BINARY data.
 * It follows the PEM encoding formula.
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @since   prototype v1.0
 */
public class Base64 {

    /**
     * Maps binary to char.
     */
    private static int map(int c) {
        if (c >= 0 && c <= 25) return(c + 'A');
        if (c >= 26 && c <= 51) return(c - 26 + 'a');
        if (c >= 52 && c <= 61) return(c - 52 + '0');
        if (c == 62) return('+');
        if (c == 63) return('/');
        else return(-1);
    }

    /**
     * Maps char to binary.
     */
    private static int unmap(int c) {
        if (c >= 'A' && c <= 'Z') return(c - 'A');
        if (c >= 'a' && c <= 'z') return(c + 26 - 'a');
        if (c >= '0' && c <= '9') return(c + 52 - '0');
        if (c == '+') return(62);
        if (c == '/') return(63);
        else return(-1);
    }

    /**
     * Decodes base64.
     */
    public static byte[] decode(byte[] string) {
        int length, i, j;
        byte[] buf = {0};

        if (string == null) {
            return buf;
        }
        length = string.length;
        buf = new byte[((length / 4) * 3) + 1];
        for (i = j = 0; i < length; i += 4, j += 3) {
            while (i < length && string[i] != '=' &&
                   !(string[i] >= 'a' && string[i] <= 'z') &&
                   !(string[i] >= 'A' && string[i] <= 'Z') &&
                   !(string[i] >= '0' && string[i] <= '9')) i++;
            buf[j] = (byte) ((unmap(string[i]) & 0x3f) << 2);
            if (string[i+1] == '=') {
```

```
                buf[j+1] = 0;
                break;
            }
            buf[j] |= (byte) ((unmap(string[i+1]) & 0x30) >> 4);
            buf[j+1] = (byte) ((unmap(string[i+1]) & 0x0f) << 4);
            if (string[i+2] == '=') {
                buf[j+2] = 0;
                break;
            }
            buf[j+1] |= (byte) ((unmap(string[i+2]) & 0x3c) >> 2);
            buf[j+2] = (byte) ((unmap(string[i+2]) & 0x03) << 6);
            if (string[i+3] == '=') {
                buf[j+3] = 0;
                break;
            }
            buf[j+2] |= (byte) unmap(string[i+3]) & 0x3f;
        }
        return buf;
    }

    /**
     * Encodes base64.
     */
    public static byte[] encode(byte[] bin) {
        int BLOCKS_PER_LINE = 18;
        int length, i, j, cnt;
        byte[] buf = {0};

        if (bin == null) {
            return buf;
        }
        length = bin.length;
        byte[] string = new byte[length + 2];
        System.arraycopy(bin, 0, string, 0, length);
        cnt = (((length + 3) / 3) * 4) + 2;
        cnt += (cnt / (BLOCKS_PER_LINE * 4));
        buf = new byte[cnt];
        for (i = j = cnt = 0; i < length + 3; i += 3,j += 4) {
            buf[j] = (byte) map((string[i] & 0xfc) >> 2);
            buf[j+1] = (byte) (map(((string[i] & 0x03) << 4) |
                               ((string[i+1] & 0xf0) >> 4)));
            if (string[i+1] == 0) {
                buf[j+2] = buf[j+3] = (byte) '=';
                j += 4;
                break;
            }
            buf[j+2] = (byte) (map(((string[i+1] & 0x0f) << 2) |
                               ((string[i+2] & 0xc0) >> 6)));
            if (string[i+2] == 0) {
                buf[j+3] = (byte) '=';
                j += 4;
                break;
            }
            buf[j+3] = (byte) map(string[i+2] & 0x3f);
            if (cnt >= (BLOCKS_PER_LINE - 1)) {
                buf[j+4] = (byte) '\n';
                buf[j+5] = (byte) ' ';
                j += 2;
                cnt = 0;
            }
            else cnt++;
        }
        return buf;
    }
```

```
    /**
     * Test function.
     */
    public static void main(String args[]) {
//        byte[] in = {(byte)0xff, (byte)0x33, (byte)0x55};
//        byte[] out = encode(in);
        byte[] out = encode(args[0].getBytes());
        System.out.println(new String(out));
        System.out.println(new String(decode(out)));
    }
}
```

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Session Description (SDP) Class]
 *
 * $<marconi.util.sd.>SDPPacket.java ~v1.0(prototype version), 1999/03/15 $
 * @jdk1.2, ~riK.
 */
package marconi.util.sd;

import java.util.StringTokenizer;
import marconi.util.Vector2;

/**
 * This class encapsulates and parses the <code>media</code> field of Session
 * Description Protocol.
 * <p>
 *
 * @author  ~riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.sd.SDPPacket
 * @since   prototype v1.0
 */
public class SDPMedia {

    /**
     * The session name.
     */
    protected String media = null;

    /**
     * The session information.
     */
    public String title = null;

    /**
     * The media connection information (multiple specification not supported).
     */
    public SDPConnection connection = null;

    /**
     * The bandwidth information.
     */
    public String bandwidth = null;

    /**
     * The encryption key.
     */
    public String key = null;

    /**
     * The media attributes.
     */
    public String attribute[] = null;

    /**
     * The media type (audio, video, application, ...etc.).
     */
    private String mediaType = null;

    /**
     * The transport port to which the media stream will be sent.
     */
    private String mediaPort = null;

    /**
```

```
     * The transport protocol.
     */
    private String mediaTransport = null;

    /**
     * The media formats.
     */
    private String[] mediaFormat = null;

    /**
     * The default constructor with specific format list.
     */
    public SDPMedia(String type, String port, String transport, String[] format) {
        mediaType = type;
        mediaPort = port;
        mediaTransport = transport;
        mediaFormat = format;
        media = compose();
    }

    /**
     * The default constructor with default media format.
     */
    public SDPMedia(String type, String port, String transport) {
        String[] fmt = {"0"};
        mediaType = type;
        mediaPort = port;
        mediaTransport = transport;
        mediaFormat = fmt;
        media = compose();
    }

    /**
     * The constructor.
     */
    public SDPMedia(String media) {
        this.media = media;
        Vector2 temp_vec = new Vector2();
        StringTokenizer st = new StringTokenizer(media, " ");

        if (st.hasMoreTokens()) {
            this.mediaType = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.mediaPort = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.mediaTransport = st.nextToken();
        }
        while (st.hasMoreTokens()) {
            temp_vec.addElement(st.nextToken());
        }
        this.mediaFormat = temp_vec.toStringArray();
    }

    /**
     * Returns the type of media. This variable is not directly accessible
     * because they are read-only.
     */
    public String getType() {
        return mediaType;
    }

    /**
```

```
     * Returns the port number for media communication. This
     * directly accessible because they are read-only.
     */
    public String getPort() {
        return mediaPort;
    }

    /**
     * Returns the transport protocol used. This variable is not directly
     * accessible because they are read-only.
     */
    public String getTransport() {
        return mediaTransport;
    }

    /**
     * Returns the media format list. This variable is not directly accessible
     * because they are read-only.
     */
    public String[] getFormat() {
        return mediaFormat;
    }

    /**
     * Returns concatenated media description.
     */
    protected String compose() {
        String temp_str = "";
        for (int i = 0; i < mediaFormat.length; i++) {
            temp_str = temp_str + ((i==0) ? "" : " ") + mediaFormat[i];
        }

        return mediaType + " " + mediaPort + " " + mediaTransport + " " + temp_str;
    }
}
```

SDPConnection.java Jun 17 14:32:40 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Session Description (SDP) Class]
 *
 * $<marconi.util.sd.>SDPPacket.java -v1.0(prototype version), 1999/03/15 $
 * @jdk1.2, -riK.
 */
package marconi.util.sd;

import java.util.*;

/**
 * This class encapsulates and parses the <code>connection</code> field of Session
 * Description Protocol. Currently only IP4 address type is defined by the
 * protocol.
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.sd.SDPPacket
 * @since   prototype v1.0
 */
public class SDPConnection {

    /**
     * The type of network.
     */
    public String network_type = "IN";

    /**
     * The type of address.
     */
    public String address_type = "IP4";

    /**
     * The base multicast address for connection.
     */
    public String address = "";

    /**
     * The connection ttl (time to live).
     */
    public String ttl = "";

    /**
     * The number of contiguous addresses including the base address.
     */
    public String count = "";

    /**
     * Default constructor. The subfields take the default values. It is the
     * user's responsibility to make sure that the required subfields are
     * properly initialized.
     */
    public SDPConnection() {
        super();
    }

    /**
     * Constructor. It does not check for the valid number of subfields. This
     * simple parser uses the default values if the input is short of fields.
     */
    public SDPConnection(String connection) {
        StringTokenizer st = new StringTokenizer(connection, " ");
```

SDPConnection.java Thu Jun 17 14:32:40 1999

```
        if (st.hasMoreTokens()) {
            this.network_type = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.address_type = st.nextToken();
        }
        if (address_type.startsWith("IP4")) {
            if (st.hasMoreTokens()) {
                String connectionAddress = st.nextToken();
                StringTokenizer st2 = new StringTokenizer(connectionAddress, "/");
                if (st2.hasMoreTokens()) {
                    this.address = st2.nextToken();
                }
                if (st2.hasMoreTokens()) {
                    this.ttl = st2.nextToken();
                }
                if (st2.hasMoreTokens()) {
                    this.count = st2.nextToken();
                }
            }
        }
        else {
            while (st.hasMoreTokens()) {
                this.address = address + " " + st.nextToken();
            }
            this.address = address.trim();
        }
    }

    /**
     * Returns the <code>String</code> of concatenated connection subfields.
     */
    protected String compose() {
        return network_type + " " + address_type + " " + address
               + (address_type.equals("IP4") ? "/" + ttl + "/" + count : "");
    }
}
```

```
/*
 * marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : (Session Announcement Protocol Class)
 *
 * $<marconi.util.sd.>SAPPacket.java -v1.0(prototype version), 1999/03/15 $
 * @jdk1.2, -riK.
 */
package marconi.util.sd;

import marconi.util.Timestamp;

/**
 * This class encapsulates and parses the Session Announcement Protocol (SAP)
 * Packet. Within it, SDP packet in also encapsulated. Many of the security
 * related fields are still being defined and thus not fully supported in this
 * version. It merely supports the parsing of these fields. For example, the
 * encrypted payload is not parsed and left to be handled by the next version,
 * or an object that extends this, or an object that encapsulates this (parent
 * object).
 * <p>
 * Included in the packet are the following fields:
 * <ul>
 * <li>version
 * <li>message type
 * <li>encryption bit
 * <li>compression bit
 * <li>authentication header length
 * <li>message id hash
 * <li>originating source
 * <li>authentication header
 * <li>timeout
 * <li>payload
 * </ul>
 * <p>
 *
 * @author -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.sd.MalformedSDException
 * @see     marconi.util.sd.SDPPacket
 * @since   prototype v1.0
 */
public class SAPPacket implements java.io.Serializable {
    final static String obj_name = "marconi.util.SAPPacket";

    /**
     * The version ID.
     */
    public final static int version = 1;

    /**
     * The message type.
     */
    public byte message_type = MT_ANNOUNCE;

    /**
     * The encryption bit.
     */
    public boolean encryption = false;

    /**
     * The compression bit.
     */
    public boolean compression = false;

    /**
```

```
     * The authentication header length.
     */
    public short auth_headerlen = 0;

    /**
     * The message identifier hash.
     */
    public short msgid_hash = 0;

    /**
     * The orginating source.
     */
    public byte[] source = new byte[4];

    /**
     * The authentication header.
     */
    public int[] auth_header = null;

    /**
     * The timeout.
     */
    public int timeout = 0;

    /**
     * The text payload (if encrypted contains the privacy header field).
     */
    public byte[] payload = null;

    /**
     * The date and time of the packet created.
     */
    public long timeStamp = -1;

    /**
     * The maximum payload buffer length (currently 1K bytes as limited by SAP).
     */
    public final static int MAX_BUFLEN = 1024;

    /**
     * The <code>announce</code> message type.
     */
    public final static int MT_ANNOUNCE = 0;

    /**
     * The <code>delete</code> message type.
     */
    public final static int MT_DELETE = 1;

    /**
     * Raw data containing the byte-array representation of the packet.
     */
    private byte[] data = null;

    /**
     * Creates an empty SAP packet.
     */
    public SAPPacket() {
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Creates SAP packet from a byte array.
     *
```

```
     * @param   buf     the byte array that contains SAP/SDP
     */
    public SAPPacket(byte[] buf) throws MalformedSDException {
        parse(buf);
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Obtains a SAP packet by parsing a byte array.
     *
     * @param   buf     a byte array that contains SAP/SDP.
     */
    public void parse(byte[] buf) throws MalformedSDException {
        int b0, b1, b2, b3, i = 0;

        // main sap header
        b0 = buf[i];
        b1 = (buf[i+1] << 24) >>> 24;
        b2 = (buf[i+2] << 24) >>> 24;
        b3 = (buf[i+3] << 24) >>> 24;
        int main_header = b0 << 24 | b1 << 16 | b2 << 8 | b3;
        i += 4;

        if (version != ((main_header & 0xe0000000) >>> 29)) {
            throw new MalformedSDException
                (obj_name + ".parse: incompatible version received.");
        }
        this.message_type = (byte) ((main_header & 0x1c000000) >>> 26);
        this.encryption = (main_header & 0x02000000) > 0;
        this.compression = (main_header & 0x01000000) > 0;
        this.auth_headerlen = (short) ((main_header & 0x00ff0000) >>> 16);
        this.msgid_hash = (short) (main_header & 0x0000ffff);

        // originating source
        this.source[0] = buf[i];
        this.source[1] = buf[i+1];
        this.source[2] = buf[i+2];
        this.source[3] = buf[i+3];
        i += 4;

        // authentication header
        this.auth_header = new int[auth_headerlen];
        for (int j = 0; j < auth_headerlen; i += ++j*4) {
            b0 = buf[i];
            b1 = (buf[i+1] << 24) >>> 24;
            b2 = (buf[i+2] << 24) >>> 24;
            b3 = (buf[i+3] << 24) >>> 24;
            this.auth_header[j] = b0 << 24 | b1 << 16 | b2 << 8 | b3;
        }

        // 32-bit timeout field
        if (encryption) {
            b0 = buf[i];
            b1 = (buf[i+1] << 24) >>> 24;
            b2 = (buf[i+2] << 24) >>> 24;
            b3 = (buf[i+3] << 24) >>> 24;
            this.timeout = b0 << 24 | b1 << 16 | b2 << 8 | b3;
            i += 4;
        }

        // payload
        int payloadlen = Math.min(MAX_BUFLEN, buf.length - i);
        this.payload = new byte[payloadlen];
        System.arraycopy(buf, i, this.payload, 0, payloadlen);
```

```
    }

    /**
     * Returns the buffer that represents this SAP packet.
     *
     * @return  byte array representation of this SAP packet (including payload).
     */
    public byte[] compose() {
        int b0, b1, b2, b3, i, j;

        // integrity check
        if (payload == null) {
            return null;
        }
        if (auth_header != null) {
            // just in case they don't match
            auth_headerlen = (short) auth_header.length;
        }
        int length = 4 + 4 + auth_headerlen * 4 +
                     ((encryption) ? 4 : 0) + payload.length;
        data = new byte[length];

        // main header
        b0 = version;
        b1 = message_type;
        b2 = (encryption) ? 1 : 0;
        b3 = (compression) ? 1 : 0;
        data[0] = (byte) (b0 << 5 | b1 << 2 | b2 << 1 | b3);
        data[1] = (byte) (auth_headerlen);
        data[2] = (byte) (msgid_hash >> 8);
        data[3] = (byte) (msgid_hash);
        data[4] = source[0];
        data[5] = source[1];
        data[6] = source[2];
        data[7] = source[3];

        // authentication header
        for (j = 0, i = 8; j < auth_headerlen; i += ++j*4) {
            data[i] = (byte) (auth_header[j] >> 24);
            data[i+1] = (byte) (auth_header[j] >> 16);
            data[i+2] = (byte) (auth_header[j] >> 8);
            data[i+3] = (byte) (auth_header[j]);
        }

        // timeout field
        if (encryption) {
            data[i] = (byte) (timeout >> 24);
            data[i+1] = (byte) (timeout >> 16);
            data[i+2] = (byte) (timeout >> 8);
            data[i+3] = (byte) (timeout);
            i += 4;
        }

        // payload
        System.arraycopy(payload, 0, data, i, payload.length);

        return data;
    }
```

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Session Description (SDP) Class]
 *
 * $<marconi.util.sd.>SDPOrigin.java -v1.0(prototype version), 1999/04/06 $
 * @jdk1.2, -riK.
 */
package marconi.util.sd;

import java.util.StringTokenizer;

/**
 * This class encapsulates and parses the <code>origin</code> field of Session
 * Description Protocol.
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.sd.SDPPacket
 * @since   prototype v1.0
 */
public class SDPOrigin {

    /**
     * The username.
     */
    public String username = "-";

    /**
     * The session id.
     */
    public String session_id = "0";

    /**
     * The announcement version.
     */
    public String version = "0";

    /**
     * The type of network.
     */
    public String network_type = "IN";

    /**
     * The type of address.
     */
    public String address_type = "IP4";

    /**
     * The originating network address.
     */
    public String address = "";

    /**
     * Default constructor. The subfields take the default values. It is the
     * user's responsibility to make sure that the required subfields are
     * properly initialized.
     */
    public SDPOrigin() {
        super();
    }

    /**
     * Constructor. It does not check for the valid number of subfields. This
     * simple parser uses the default values if the input is short of fields.
```

SDPOrigin.java Jun 17 14:33:06 1999 2

```
     */
    public SDPOrigin(String origin) {
        StringTokenizer st = new StringTokenizer(origin, " ");

        if (st.hasMoreTokens()) {
            this.username = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.session_id = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.version = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.network_type = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.address_type = st.nextToken();
        }
        if (st.hasMoreTokens()) {
            this.address = st.nextToken();
        }
    }

    /**
     * Returns the <code>String</code> of concatenated origin subfields.
     */
    protected String compose() {
        return username + " " + session_id + " " + version + " "
                + network_type + " " + address_type + " " + address;
    }
```

SDPPacket.java Thu 17 4:33:15 1999 1

```
/* marconiNet - Internet Radio Network
 * Distributed Radio Antenna Server (RAS) : [Session Description (SDP) Class]
 *
 * S<marconi.util.sd.>SDPPacket.java -v1.0(prototype version), 1999/03/15 $
 * @jdk1.2, -riK.
 */
package marconi.util.sd;

import java.util.*;
import java.io.*;
import marconi.util.Timestamp;
import marconi.util.Vector2;

/**
 * This class encapsulates and parses the Session Description Protocol (SDP)
 * Packet. It does not support parsing of the full SDP fields but only enough
 * to support CDP (Channel Description Protocol). Thus, it does not parse into
 * the subfields of SDP if they are not to be used for channel description.
 * Further, the parser tolerates other passive errors such as the existence of
 * fields that are not defined. However, the composing and parsing of SDP is
 * 100% compliant with the standard specification. In order to facilitate this
 * the parser strictly checks for the order and the existence of the required
 * fields.
 * <p>
 * The syntax of the SDP fields are given below:
 * <ul>
 * <li>v=0
 * <li>o=&lt;username&gt; &lt;session id&gt; &lt;version&gt; &lt;network type&gt;
 * &lt;address type&gt; &lt;address&gt;                                          * *
 * <li>s=&lt;session name&gt;                                                   * *
 * <li>i=&lt;session description&gt;
 * <li>u=&lt;uri&gt;
 * <li>e=&lt;email address&gt;
 * <li>p=&lt;phone number&gt;
 * <li>c=&lt;network type&gt; &lt;address type&gt; &lt;connection address&gt;
 * <li>b=&lt;modifier&gt;:&lt;bandwidth-value&gt;
 * <li>t=&lt;start time&gt; &lt;stop time&gt;
 * <li>r=&lt;repeat interval&gt; &lt;active duration&gt; &lt;list of offsets&gt;
 * <li>z=&lt;adjustment time&gt; &lt;offset&gt; ...
 * <li>k=&lt;method&gt;
 * <li>k=&lt;method&gt;:&lt;encryption key&gt;
 * <li>a=&lt;attribute&gt;
 * <li>a=&lt;attribute&gt;:&lt;value&gt;
 * <li>m=&lt;media&gt; &lt;port&gt; &lt;transport&gt; &lt;fmt list&gt;
 * </ul>
 * <p>
 *
 * @author  -riK.
 * @version $Revision: 1.0 $
 * @see     marconi.util.sd.MalformedSDException
 * @see     marconi.util.sd.SDPOrigin
 * @see     marconi.util.sd.SDPMedia
 * @see     marconi.util.sd.SDPConnection
 * @see     marconi.util.sd.SAPPacket
 * @since   prototype v1.0
 */
public class SDPPacket implements java.io.Serializable {
    final static String obj_name = "marconi.util.SDPPacket";

    /**
     * The protocol version.
     */
    public final static int version = 0;
```

SDPPacket.java [redacted] Jun 17 14:33:15 1999

```
    /**
     * The owner/creator and session identifier.
     */
    public SDPOrigin origin = new SDPOrigin();

    /**
     * The session name.
     */
    public String name = "";

    /**
     * The session information.
     */
    public String info = null;

    /**
     * The URI of description.
     */
    public String uri = null;

    /**
     * The contact email address.
     */
    public String[] email = null;

    /**
     * The contact phone number.
     */
    public String[] phone = null;

    /**
     * The session level connection information.
     */
    public SDPConnection connection = new SDPConnection();

    /**
     * The bandwidth information.
     */
    public String bandwidth = null;

    /**
     * The time zone adjustments.
     */
    public String zone = null;

    /**
     * The encryption key.
     */
    public String key = null;

    /**
     * The session attributes.
     */
    public String[] attribute = null;

    /**
     * The active times (NTP in seconds - 0h on January 1900).
     */
    public String[] active = {"0 0"};

    /**
     * The repeat times.
     */
    public String[] repeat = null;
```

```
    /**
     * The media description.
     */
    public SDPMedia[] media = new SDPMedia[MAX_MEDIACOUNT];

    /**
     * The date and time of the packet created.
     */
    public long timeStamp = -1;

    /**
     * The number of media descriptions.
     */
    public int mn = 0;

    /**
     * The maximum number of media descriptions.
     */
    public final static int MAX_MEDIACOUNT = 10;

    /**
     * Raw data containing the byte-array representation of the packet, if it has
     * one.
     */
    private byte[] data = null;

    /**
     * Creates an empty SDP packet.
     */
    public SDPPacket() {
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Creates SDP packet from a (received) SAP payload.
     *
     * @param   buf          a byte array that contains SDP.
     */
    public SDPPacket(byte[] buf) throws MalformedSDException {
        parse(buf);
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Creates a SDP packet from a file.
     *
     * @param   file    pathname to where the SDP-file is located.
     */
    public SDPPacket(String file) throws MalformedSDException {
        parse(file);
        this.timeStamp = Timestamp.get_current();
    }

    /**
     * Obtains a SDP packet by parsing the SAP payload.
     *
     * @param   buf          a byte array that contains SAP payload (SDP content).
     */
    public void parse(byte[] buf) throws MalformedSDException {

        /*
         * Open packet buffer as stream.
         */
```

SDPPacket.java Jun 17 14:33:15 1999

```
        BufferedReader buf_in = new BufferedReader(new
            (new ByteArrayInputStream(buf)));

        try {
            parse(buf_in);
            buf_in.close();
        }
        catch (MalformedSDException e) {
            throw new MalformedSDException
                (obj_name + ".parse: packet cannot be parsed.");
        }
        catch (IOException e) {
        }
    }

    /**
     * Obtains a SDP packet from a file.
     *
     * @param   file_path       name and path of the SDP announcement file.
     */
    public void parse(String file_path) throws MalformedSDException {

        /*
        * Open file.
        */
        File file = new File(file_path);
        BufferedReader file_in = null;
        try {
            file_in = new BufferedReader(new FileReader(file));
            parse(file_in);
            file_in.close();
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (MalformedSDException e) {
            throw new MalformedSDException
                (obj_name + ".parse: file cannot be parsed.");
        }
        catch (IOException e) {

        }
    }

    /**
     * Parse ordered fields.
     */
    protected void parse(BufferedReader parser) throws MalformedSDException {
        boolean perror = false;

parse_err:
        try {
            Vector2 temp_vec = new Vector2();
            String line = parser.readLine();

            // v
            if (line != null && line.startsWith("v=")) {
                String v_str = line.substring(line.indexOf('=') + 1).trim();

        System.out.println(obj_name + ".parse: " + line);

                if (Integer.parseInt(v_str) > this.version) {
                    throw new MalformedSDException
                        (obj_name + ".parse: incompatible version received.");
```

```
            }
            line = parser.readLine();
        }
        else {
            perror = true;
            break parse_err;
        }
        // o
        if (line != null && line.startsWith("o=")) {

System.out.println(obj_name + ".parse: " + line);

            this.origin = new SDPOrigin(line.substring
                (line.indexOf('=') + 1).trim());
            line = parser.readLine();
        }
        else {
            perror = true;
            break parse_err;
        }
        // s
        if (line != null && line.startsWith("s=")) {

System.out.println(obj_name + ".parse: " + line);

            this.name = line.substring(line.indexOf('=') + 1).trim();
            line = parser.readLine();
        }
        else {
            perror = true;
            break parse_err;
        }
        // i
        if (line != null && line.startsWith("i=")) {

System.out.println(obj_name + ".parse: " + line);

            this.info = line.substring(line.indexOf('=') + 1).trim();
            line = parser.readLine();
        }
        // u
        if (line != null && line.startsWith("u=")) {

System.out.println(obj_name + ".parse: " + line);

            this.uri = line.substring(line.indexOf('=') + 1).trim();
            line = parser.readLine();
        }
        // e
        if (temp_vec.size() > 0) {
            temp_vec.removeAllElements();
        }
        while (line != null && line.startsWith("e=")) {

System.out.println(obj_name + ".parse: " + line);

            temp_vec.addElement(line.substring(line.indexOf('=') + 1).trim());
            line = parser.readLine();
        }
        if (temp_vec.size() > 0) {
            this.email = temp_vec.toStringArray();
        }
        // p
        if (temp_vec.size() > 0) {
```

```
            temp_vec.removeAllElements();
        }
        while (line != null && line.startsWith("p=")) {

    System.out.println(obj_name + ".parse: " + line);

            temp_vec.addElement(line.substring(line.indexOf('=') + 1).trim());
            line = parser.readLine();
        }
        if (temp_vec.size() > 0) {
            this.phone = temp_vec.toStringArray();
        }
        // c
        if (line != null && line.startsWith("c=")) {

    System.out.println(obj_name + ".parse: " + line);

            this.connection = new SDPConnection(line.substring
                (line.indexOf('=') + 1).trim());
            line = parser.readLine();
        }
        // b
        if (line != null && line.startsWith("b=")) {

    System.out.println(obj_name + ".parse: " + line);

            this.bandwidth = line.substring(line.indexOf('=') + 1).trim();
            line = parser.readLine();
        }
        // z
        if (line != null && line.startsWith("z=")) {

    System.out.println(obj_name + ".parse: " + line);

            this.zone = line.substring(line.indexOf('=') + 1).trim();
            line = parser.readLine();
        }
        // k
        if (line != null && line.startsWith("k=")) {
            this.key = line.substring(line.indexOf('=') + 1).trim();
            line = parser.readLine();
        }
        // a
        if (temp_vec.size() > 0) {
            temp_vec.removeAllElements();
        }
        while (line != null && line.startsWith("a=")) {

    System.out.println(obj_name + ".parse: " + line);

            temp_vec.addElement(line.substring(line.indexOf('=') + 1).trim());
            line = parser.readLine();
        }
        if (temp_vec.size() > 0) {
            this.attribute = temp_vec.toStringArray();
        }
        // t
        if (temp_vec.size() > 0) {
            temp_vec.removeAllElements();
        }
        while (line != null && line.startsWith("t=")) {

    System.out.println(obj_name + ".parse: " + line);
```

```
            temp_vec.addElement(line.substring(line.index
            line = parser.readLine();
        }
        if (temp_vec.size() > 0) {
            this.active = temp_vec.toStringArray();
        }
        else {
            perror = true;
            break parse_err;
        }
        // r
        if (temp_vec.size() > 0) {
            temp_vec.removeAllElements();
        }
        while (line != null && line.startsWith("r=")) {
            temp_vec.addElement(line.substring(line.indexOf('=') + 1).trim());
            line = parser.readLine();
        }
        if (temp_vec.size() > 0) {
            this.repeat = temp_vec.toStringArray();
        }
        // m
        while (line != null && line.startsWith("m=")) {

    System.out.println(obj_name + ".parse: " + line);

            if (mn < MAX_MEDIACOUNT) {
                mn++;
                this.media[mn-1] = new SDPMedia(line.substring
                    (line.indexOf('=') + 1).trim());
                line = parser.readLine();
                // i
                if (line != null && line.startsWith("i=")) {
                    media[mn-1].title = line.substring
                        (line.indexOf('=') + 1).trim();
                    line = parser.readLine();
                }
                // c
                if (line != null && line.startsWith("c=")) {
                    media[mn-1].connection = new SDPConnection
                        (line.substring(line.indexOf('=') + 1).trim());
                    line = parser.readLine();
                }
                // b
                if (line != null && line.startsWith("b=")) {
                    media[mn-1].bandwidth = line.substring
                        (line.indexOf('=') + 1).trim();
                    line = parser.readLine();
                }
                // k
                if (line != null && line.startsWith("k=")) {
                    media[mn-1].key = line.substring
                        (line.indexOf('=') + 1).trim();
                    line = parser.readLine();
                }
                // a
                if (temp_vec.size() > 0) {
                    temp_vec.removeAllElements();
                }
                while (line != null && line.startsWith("a=")) {
                    temp_vec.addElement(line.substring
                        (line.indexOf('=') + 1).trim());
                    line = parser.readLine();
                }
```

```
                    if (temp_vec.size() > 0) {
                        media[mn-1].attribute = temp_vec.toStringArray();
                    }
                }
            }
            if (mn == 0) {
                perror = true;
                break parse_err;
            }
        }
        catch (IOException e) {
            throw new MalformedSDException
                (obj_name + ".parse: session description cannot be parsed.");
        }

        /*
        * Uncomment below if strict formating is desired.
        */
        if (perror) {
            throw new MalformedSDException
                (obj_name + ".parse: session description cannot be parsed.");
        }
    }

    /**
     * Composes a byte array that represents this SDP packet.
     * All appropriate fields, not defined as <code>null</code>, are used
     * to make up this packet. The syntax integrity check is not performed
     * during this process. The <code>MalformedSDException</code> will be
     * thrown by the parser when the receiver of this packet tries to parse
     * it.
     *
     * @return  a byte array consisting of the valid SDP fields in order.
     */
    public byte[] compose() {
        ByteArrayOutputStream ba = new ByteArrayOutputStream();
        PrintWriter ba_out = new PrintWriter(new OutputStreamWriter(ba), true);

        ba_out.println("v=" + version);
        if (origin != null) {
            ba_out.println("o=" + origin.compose());
        }
        if (name != null) {
            ba_out.println("s=" + name);
        }
        if (info != null) {
            ba_out.println("i=" + info);
        }
        if (uri != null) {
            ba_out.println("u=" + uri);
        }
        if (email != null) {
            for (int i = 0; i < email.length; i++)
                if (email[i] != null)
                    ba_out.println("e=" + email[i]);
        }
        if (phone != null) {
            for (int i = 0; i < phone.length; i++)
                if (phone[i] != null)
                    ba_out.println("p=" + phone[i]);
        }
        if (connection != null) {
            ba_out.println("c=" + connection.compose());
        }
```

```
        if (bandwidth != null) {
            ba_out.println("b=" + bandwidth);
        }
        if (zone != null) {
            ba_out.println("z=" + zone);
        }
        if (key != null) {
            ba_out.println("k=" + key);
        }
        if (attribute != null) {
            for (int i = 0; i < attribute.length; i++)
                if (attribute[i] != null)
                    ba_out.println("a=" + attribute[i]);
        }
        if (active != null) {
            for (int i = 0; i < active.length; i++)
                if (active[i] != null)
                    ba_out.println("t=" + active[i]);
        }
        if (repeat != null) {
            for (int i = 0; i < repeat.length; i++)
                if (repeat[i] != null)
                    ba_out.println("r=" + repeat[i]);
        }
        for (int i = 0; i < media.length; i++) {
            if (media[i] != null) {
                if (media[i].media != null) {
                    ba_out.println("m=" + media[i].media);
                }
                if (media[i].title != null) {
                    ba_out.println("i=" + media[i].title);
                }
                if (media[i].connection != null) {
                    ba_out.println("c=" + media[i].connection.compose());
                }
                if (media[i].bandwidth != null) {
                    ba_out.println("b=" + media[i].bandwidth);
                }
                if (media[i].key != null) {
                    ba_out.println("k=" + media[i].key);
                }
                if (media[i].attribute != null) {
                    for (int j = 0; j < media[i].attribute.length; j++)
                        if (media[i].attribute[j] != null)
                            ba_out.println("a=" + media[i].attribute[j]);
                }
            }
        }
        data = ba.toByteArray();
        ba_out.close();

        return data;
    }
```

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/16913

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : H04L 12/28; H04Q 11/04
US CL : 455/3.1, 503, 510, 515, 518, 519
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 455/3.1, 503, 510, 515, 518, 519.
370/352, 390, 392.

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

None

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST 1.1

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,572,678 A (HOMMA et al.) 05 November 1996, col. 3, lines 3-45.	27-32
Y	US 5,748,736 A (MITTRA) 05 May 1998, col. 4, line 5 to col. 5, line 55.	1-26
Y	US 5,893,091 A (HUNT et al.) 06 April 1999, col. 4, line 38 to col. 7, line 24.	1-32
Y	US 5,903,559 A (ACHARYA et al.) 11 May 1999, col. 5, lines 20-63.	1-32
Y, E	US 6,085,101 A (JAIN et al.) 04 July 2000, col. 3, line 43 to col. 5, line 40.	1-32

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 05 SEPTEMBER 2000	Date of mailing of the international search report 05 OCT 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer THUAN T. NGUYEN Telephone No. (703) 308-6860

Form PCT/ISA/210 (second sheet) (July 1998)*

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 February 2001 (08.02.2001)

PCT

(10) International Publication Number
WO 01/10146 A1

(51) International Patent Classification[7]: H04Q 7/22

(21) International Application Number: PCT/US00/21065

(22) International Filing Date: 2 August 2000 (02.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/365,963 2 August 1999 (02.08.1999) US

(71) Applicant: **QUALCOMM INCORPORATED** [US/US]; 5775 Morehouse Drive, San Diego, CA 92121-1714 (US).

(72) Inventors: **NEVO, Ron**; Mitzpe Aviv, 20187 Misgav (IL). **VAKULENKO, Michael**; Harav Ankave 22/19, 35849 Haifa (IL). **KOLOR, Sergio**; Nahshon 4/1, 34612 Haifa (IL). **NIZRI, Shlomo**; 17905 Kibutz Hasolelim (IL). **LEVY, Atai**; Oren Street 7, 34612 Haifa (IL).

(74) Agents: **WADSWORTH, Philip, R.** et al.: Qualcomm Incorporated, 5775 Morehouse Drive, San Diego, CA 92121-1714 (US).

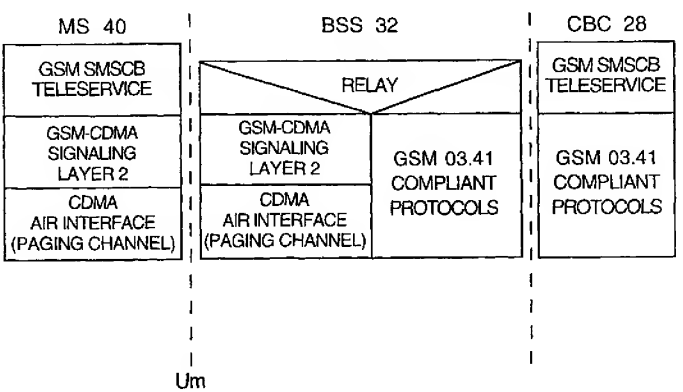
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: CELL BROADCAST IN A HYBRID GSM/CDMA NETWORK



(57) Abstract: In a GSM mobile wireless telecommunications system, a method for broadcasting messages over a CDMA air interface includes conveying a message to a base station substantially in accordance with a GSM cell broadcast service protocol, and transmitting the message to a mobile station over the CDMA air interface. The message is transmitted substantially in accordance with a CDMA transmission standard, preferably the IS-95B standard.

WO 01/10146 A1

CELL BROADCAST IN A HYBRID GSM/CDMA NETWORK

I. Field of the invention

The present invention relates generally to wireless telecommunications, and specifically to advanced cellular telephone networks.

II. Background of the invention

The Global System for Mobile (GSM) telecommunications is used in cellular telephone networks in many countries around the world. Existing GSM networks are based on time-division multiple access (TDMA) digital communications technology. GSM offers a useful range of network services and standards.

One of these GSM network services is a short message service - cell broadcast (SMSCB), for distributing short messages from a Cell Broadcast Center (CBC) via Base Station Subsystems (BSSs) in the network to subscriber units, or Mobile Stations (MSs). SMSCB messages may come from various sources, such as traffic and weather reports. These messages are broadcast to MSs in defined geographical areas, known as cell broadcast areas, over a dedicated Cell Broadcast Channel (CBCH) without requiring acknowledgment from the MS. The messages are received by the MS only in idle mode (i.e., when a telephone call is not in progress). SMSCB and related interfaces, protocol stacks and message formats are described, inter alia, in GSM standards 02.03, 03.41, 03.49 and 04.12, which are incorporated herein by reference. It is noted, however, that there is no mandatory protocol defined by G5M standards between the CBC and the BSSs. Rather, the interface protocol is left to be determined by operators of the network and cell broadcast services, based on primitives defined by the 03.41 standard.

Code-division multiple access (CDMA) is an improved digital communications technology using direct sequence spread spectrum modulation techniques, which affords more efficient use of radio bandwidth than TDMA, as well as a more reliable, fade-free link between cellular telephone subscribers and base stations. The leading CDMA standard is TIA/EIA-95, promulgated by the Telecommunications Industry Association (TIA), which is incorporated herein by reference. In the context of the present patent application and in the claims, this standard is referred to as IS-95, by which name it is commonly known in the cellular communications industry. A recent version of the standard, known as TIA/EIA-95-B (hereinafter IS-95B), has advanced networking features including

procedures for broadcast of messages over paging channels, which are monitored and received by compatible MSs.

PCT patent application PCT/US96/20764 which is based upon U.S. patent application serial no. 08/575,413 entitled "Wireless Telecommunications System Utilizing CDMA Radio Frequency Signal Modulation in Conjunction with the GSM AInterface Telecommunications Network Protocol," filed December 20, 1995, which are assigned to the assignee of the present patent application and both incorporated herein by reference, describes a wireless telecommunications system that uses a CDMA air interface (i.e., basic RF communications protocols) to implement GSM network services and protocols. Using this system, at least some of the TDMA base stations (BSSs) and subscriber units of an existing GSM network would be replaced or supplemented by corresponding CDMA equipment. CDMA BSSs in this system are adapted to communicate with GSM mobile switching centers (MSCs); via a standard GSM A-interface. The core of GSM network services is thus maintained, and the changeover from TDMA to CDMA is transparent to users.

Hybrid cellular communications networks, incorporating both GSM and CDMA elements, are also described in PCT patent publications WO 95/24771 and WO 96/21999, and in an article by Tscha, et al., entitled "A Subscriber Signaling Gateway between CDMA Mobile Station and GSM Mobile Switching Center," in Proceedings of the 2nd International Conference on Universal Personal Communications, Ottawa (1993), pp. 181-185, which are incorporated herein by reference. None of these publications deal with issues of support of short message or cell broadcast services, such as SMSCB, in such hybrid networks.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide methods and apparatus for use in a mixed GSM/CDMA cellular communications network.

It is a further object of some aspects of the present invention to provide methods and apparatus enabling realization of GSM Cell Broadcast Service over a CDMA air interface.

In preferred embodiments of the present invention, a mixed GSM/CDMA cellular communications system includes both TDMA and CDMA base stations (BSSs), which are in contact with a Cell Broadcast Center (CBC) based on GSM standards. Systems of this type are described generally in U.S. patent application serial no. 09/119,717 entitled "Base Station Handover in a

Hybrid GSM/CDMA Network," U.S. Patent Application Serial No. 09/119,717, filed July 20, 1998, which is assigned to the assignee of the present patent application and is incorporated herein by reference. A subscriber unit in the network, referred to herein as a mobile station (MS), is preferably capable of communicating with both types of base stations by appropriately switching between TDMA add CDMA air interfaces. GSM network protocols are used over both types of air interface, so that the MS receives short messages broadcast from the GSM CBC whether the MS is in communication with a TDMA BSS or a CDMA BSS. The present invention thus enables CDMA BSSs and MSs to be integrated into a GSM network infrastructure without compromising the ability of the CBC associated with the network to broadcast messages based on GSM standards to substantially any MS, and with substantially no other modification required to existing infrastructure.

In some preferred embodiments of the present invention, GSM short message service - cell broadcast (SMSCB) messages are broadcast to the MS over the CDMA air interface using a CDMA paging channel. Preferably, the messages are broadcast in accordance with broadcast procedures over paging channels defined by the IS-95B standard.

There is therefore provided, in accordance with a preferred embodiment of the present invention, in a GSM mobile wireless telecommunications system a method for broadcasting messages over a CDMA air interface, including:

conveying a message to a base station substantially in accordance with a GSM cell broadcast service protocol; and

transmitting the message to a mobile station over the CDMA air interface.

Preferably, transmitting the message includes transmitting a message substantially in accordance with a CDMA transmission standard, most preferably IS95B.

Preferably, transmitting the message includes transmitting a message over a paging channel, wherein transmitting the message includes directing the message to one or more specified cells.

In a preferred embodiment, transmitting the message includes transmitting a message so as to have a high likelihood of being received by the mobile station while the mobile station is operating in a slotted mode, most preferably by transmitting a broadcast page followed by transmission of a message. Preferably, transmitting the broadcast page includes transmitting a page in a predetermined slot monitored by the mobile station in a periodic broadcast paging cycle. Optionally, the method includes transmitting a schedule

message including information regarding one or more broadcast pages to be transmitted in a schedule period including one or more periodic broadcast paging cycles.

Preferably, transmitting the message includes transmitting a single message multiple times, substantially in accordance with a GSM cell broadcast command, wherein transmitting the single message includes assigning a broadcast address to the message, such that when the mobile station receives a second message having the same broadcast address as a first, earlier message, the second message is discarded. Preferably, transmitting the single message multiple times includes repeating transmission of the message at a repetition frequency determined responsive to the cell broadcast command.

Further preferably, transmitting the message includes encapsulating a GSM cell broadcast message in an IS-95 message.

Preferably, conveying the message includes receiving a message from a cell broadcast center.

In a preferred embodiment, transmitting the message includes transmitting a message header including a message identifier field indicative of a characteristic of the message, and the mobile station determines whether to decode or discard the message responsive to the characteristic.

There is also provided, in accordance with a preferred embodiment of the present invention, apparatus for broadcasting short messages from a cell broadcast center over a CDMA air interface, including a base station subsystem, which receives the messages from the cell broadcast center substantially in accordance with a GSM cell broadcast service protocol and transmits the message to a mobile station over the CDMA air interface.

Preferably, the base station subsystem transmits the message substantially in accordance with a CDMA transmission standard, most preferably IS-95B.

Preferably, the base station subsystem transmits the message over a paging channel and directs the message to one or more specified cells.

Preferably, the base station subsystem transmits the message so as to have a high likelihood of being received by the mobile station while the mobile station is operating in a slotted mode, wherein before transmitting the message, the base station subsystem transmits a page in a predetermined slot monitored by the mobile station in a periodic broadcast paging cycle.

Further preferably, the base station subsystem transmits a single message multiple times, substantially in accordance with a GSM cell broadcast command, wherein the base station subsystem repeats transmission of the message at a

repetition frequency determined responsive to the cell broadcast command. Further preferably, the base station subsystem assigns a broadcast address to the message, such that when the mobile station receives a second message having the same broadcast address as a first, earlier message, the second message is discarded.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more fully understood from the following detailed description of the preferred embodiments thereof, taken together with the drawings in which:

Fig. 1 is a schematic block diagram of a hybrid GSM/CDMA cellular communications system, in accordance with a preferred embodiment of the present invention;

Fig. 2 is a schematic block diagram illustrating communications protocols between elements of the system of Fig. 1, in accordance with a preferred embodiment of the present invention; and

Fig. 3 is a schematic block diagram illustrating a message format for use in cell broadcast service messages, in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Reference is now made to Fig. 1, which is a schematic block diagram of a hybrid GSIVI/CDMA cellular communications system 20, in accordance with a preferred embodiment of the present invention. System 20 is built around a public land mobile network (PLMN) 22, which is based on the GSM communications standard, as is known in the art and described briefly hereinabove. Infrastructure for such networks already exists and is in wide use in many countries, and the present invention has the advantage of enabling gradual introduction of CDMA service in conjunction with such a network without requiring major changes to the existing switching infrastructure.

PLMN 22 comprises at least one mobile-services switching center (MSC) 24, or possibly a number of such centers (although only one MSC is shown here for clarity of illustration), which controls network operations within a geographical area. Among other functions, MSC 24 is responsible for location registration of subscriber units and handover of subscriber units between base

stations, as well as linking PLMN 22 to a public switched telephone network (PSTN) and/or packet data network (PDN) 48. The PLMN also comprises a network management center (NMC) 26 and a cell broadcast center (CBC) 28. Other aspects of system 20 and details regarding a mobile station (MS) 40 in the system, are described further in the above-mentioned U.S. and PCT Patent Applications.

System 20 includes a plurality of MSs 40, which communicate with PLMN 22 via a plurality of base station subsystems (BSS) 30 and 32 over a wireless RF link at one or more of the accepted cellular communications frequencies. MS 40, which is also known as a subscriber unit, is preferably capable of communicating with both GSM BSS 30, using a standard GSM TDMA radio communications protocol, and CDMA BSS 32, using CDMA-based communication methods described hereinbelow. Although for the sake of clarity, only one each of MS 40, GSM BSS 30 and CDMA BSS 32 is shown in Fig. 1, it will be understood that in actuality, system 20 typically comprises a plurality of each of these system elements.

Communications between CDMA BSS 32 and MS 40 use a CDMA radio "air interface," which is preferably based on the IS-95 standard for CDMA communications, and most preferably with the TIA/EIA-95-B version of the standard, which is incorporated herein by reference, with necessary modifications as described herein. BSS 32 is built around a base station controller (BSC) 34, which controls and communicates with a number of base station transceivers (BTS) 36. Each BTS transmits RF signals to and receives RF signals from MS 40 when the MS is within a geographical area, or cell, served by the particular BTS. On the other hand, when MS 40 is within a cell served by GSM BSS 30, the MS preferably communicates with BSS 30 over a GSM/TDMA air interface to GSM/TDMA BTSs (not shown in the figure for the sake of simplicity).

Both GSM BSS 30 and CDMA BSS 32 communicate with and are controlled by MSC 24, substantially in accordance with GSM standards, i.e., via the GSM standard A-interface, as further described in the above-mentioned U.S. and PCT Patent Applications. BSS 32 also comprises a radio operation and maintenance center (OMCR) 38, which communicates with NMC 26 over a GSM-standard Q3 interface.

BSC 34 communicates with CBC 28 so as to receive short messages to be broadcast over the air, based on the above-mentioned GSM SMSCB standards. One CBC 28 is typically connected to BSCs of a plurality of BSSs, which may include both CDMA BSSs, such as BSS 32, and GSM/TDMA BSSs, such as BSS

30. The CBC may receive short messages for broadcast from several cell broadcast entities (not shown), such as weather and traffic report centers, as described in GSM specifications.

CBC 28 is responsible for management of cell broadcast short messages and issuing commands to BSC 34 (and other BSCs), including:

- Allocation of serial numbers for identification of messages, as specified by GSM standard 03.41. The serial number is included in the header of each message (described further hereinbelow with reference to Fig. 3). It typically includes fields specifying a geographical scope of the message, a message code for differentiating between messages from the same source and of the same type, and an update number.
- Modifying or deleting messages held by the BSC.
- Initiating broadcast by sending cell broadcast messages to the BSC, and where necessary padding the messages to a length of 82 octets, based on GSM standard 03.41.
- Determining a set of cells or BTSs to which the message should be broadcast and indicating the geographical scope of the message. A list of the determined cells or BTSs is conveyed to the BSC, which then distributes the broadcast accordingly.
- Determining a time at which the message should commence being broadcast.
- Determining a time at which the message should cease being broadcast, and instructing the BSC accordingly.
- Determining a rate at which the message broadcast should be repeated.
- Determining the cell broadcast channel on which the message should be broadcast.

BSC 34 performs the following function vis-a-vis CBC 28:

- Receiving and interpreting cell broadcast commands.
- Storing cell broadcast messages.
- Scheduling of the messages over CDMA paging channels, as described further hereinbelow.
- Providing an indication to the CBC when the desired message repetition rate cannot be achieved, for example, when the desired rate is too high.
- Acknowledging successful execution of CBC commands.
- Reporting to the CBC when a command cannot be understood or cannot be executed.

- Routing cell broadcast messages to the appropriate BTSs, preferably using directed paging procedures, as described further hereinbelow.
- Transferring cell broadcast service (CBS) information to each BTS.
- Optionally, generating schedule messages, as described further hereinbelow, indicating an intended schedule of transmissions.

Preferably, when a message conveyed from BSC 34 to CBC 28 pertains to multiple cells, the message is conveyed only once, together with a list of the cells to which it applies.

MS 40 comprises mobile equipment (ME) 42, which preferably includes either two radio transceivers, one configured for TDMA operation and one for CDMA, or a single transceiver which can dynamically switch between TDMA and CDMA. The MS includes mobile termination (MT), which supports terminal equipment (TE) 46 for voice and data input and output. In addition, MS 40 comprises a subscriber identity module (SIM) 44, in accordance with GSM standards, which is used in authenticating the identity of a user of MS 40 in a manner substantially transparent to and independent of the CDMA air interface. Although preferred embodiments are described herein with reference to MS 40 having dual CDMA/TDMA air interface compatibility, it will be understood that the principles of the present invention may similarly be applied to systems using mobile stations having only CDMA compatibility or, mutatis mutandis, to other systems using GSM networking standards.

MS 40 receives the cell broadcast messages transmitted by BTS 36 from BSC 34. Typically, TE 46 is used to display the messages in alphanumeric format, as is known in the art. In addition, MS 40 has the following capabilities with regard to the 25 cell broadcast messages:

- Identifying and discarding messages that have a message identifier indicating that the subject matter of the message is not of interest to a subscriber using the MS.
- Ignoring repeated broadcast of messages already received (i.e., a message broadcast address has not changed).
- Transferring the message via a R-interface between ME 42 and TE 46, when such an interface is supported.

Reference is now made to Fig. 2, which is a schematic block diagram showing protocol stacks used in conveying SMSCB teleservice messages between CBC 28 and MS 40 via BSS 32, in accordance with a preferred embodiment of the present invention. The protocols shown in the figure enables

the substantially unmodified GSM CBC to convey short messages to MS 40, based on GSM SMSCB standards, using paging channel broadcast procedures between the BSS and the MS specified by the IS95B standard.

Cell broadcast messages are conveyed from CBC 28 to BSC 34 over a CBC-BSC interface, which is generally in accordance with GSM specifications. As noted hereinabove, the protocol to be implemented in this interface is a mattter to be agreed upon between operators of the CBC and PLMN 22, based on primitives defined in GSM standard 03.41. Preferably, as illustrated in Fig. 2, the stack is in accordance with one of the sample protocol stacks provided in GSM standard 03.49.

The GSM primitives defined by the 03.41 standard are generally received and interpreted by BSC 34 in conformity with the standard, but there are a few exceptions necessitated in order to accommodate IS-95B broadcast procedures over the paging channel. GSM standards support two kinds of cell broadcast channels (CBCH): a basic and an extended channel; but both of these channels are mapped to the same CDMA paging channel. Furthermore, whereas GSM specifications define the maximum 20 repetition frequency of a cell broadcast message as once in every 51 x 8 TDMA frames (1.883 sec), the maximum repetition frequency over the CDMA air interface, in accordance with IS-95B, is once in every periodic broadcast paging cycle, corresponding to about 2.8 sec at a minimum. BSS 32 determines a BCAST_INDEX parameter between 1 and 7, as specified by IS-95B, to set the duration of the periodic broadcast paging cycles. In the event of a conflict, such as more than one message to broadcast at a given time, BSS 32 determines the order of broadcast of the messages in accordance with predetermined criteria.

BSS 32 manages end-to-end delivery of the messages and provides synchronization and scheduling services based on the interface primitives used in the CBC-BSC interface. IS-95B includes several different paging methods, which are applicable in corresponding preferred embodiments of the present intention and are described further hereinbelow. Preferably, BSC 34 is capable of sending broadcast pages (as described hereinbelow) and messages to specific cells, so that the cell broadcast messages can be distributed to specified geographical areas.

MS 40 exchanges signals with CDMA BSS 32 over a CDMA Um interface, wherein the MS and BSS protocol stacks are modified to accommodate GSM network services, such as SMSCB, as described herein. The SMSCB messages are transmitted from BTS 36 to MS 40 over the paging channel of the CDMA air interface.

Fig. 3 is a block diagram illustrating a SMSCB message 80 transmitted over the paging channel to MS 40, in accordance with a preferred embodiment of the present invention. Message 80 has the form of a standard IS-95 Data Burst Message (DBM). Message 80 encapsulates an entire GSM cell broadcast message, including a GSM SMSCB header 86 and data 88, preferably in accordance with GSM standard 03.41. Message 80 also includes a MSG LEN field 82, set by BSS 32 to specify the total length of the message, and a CRC field 92, both in accordance with IS-95 specifications, along with a DBM header 84 and a reserved field 90 of 5 bits.

DBM header 84 is set by BSS 32 to be generally in accordance with IS-95B specifications for paging channel messages, with the exception of a broadcast address field in the DBM header, whose content is preferably set to accommodate GSM cell broadcast service, as defined in particular by GSM standard 03.41, section 9.3, "Message Format on BTS-MS interface." The broadcast address field contains a part of a cell broadcast message header generated by CBC 28, which uniquely identifies the message so that MS 40 can detect and avoid decoding messages that it has already received or which are not of interest to the subscriber. (The CBC message header includes the first six octets of the 88 octets of the complete message.) MS 40 ignores messages whose BURST_TYPE and broadcast address fields in DBM header 84 are identical to those of an earlier message already received in a given periodic broadcast slot cycle.

The structure of the broadcast address field is illustrated in Table I below:

TABLE I

Sub-Field	Length (bits)	Value
Message Type	8	0 - normal message 1 - schedule message (as described further hereinbelow) Other - reserved
Serial Number	16	For any given Message Identifier, the Serial Number is updated every time a new message is transmitted, to distinguish different messages with a common source and type.

Message Identifier	16	Identifies the source and type of the message.
Data Coding Scheme	8	Indicates intended handling of the message at the MS, including alphabet/coding and language as defined by GSM standard 03.38.

The structure of the Serial Number sub-field is also defined in accordance with the above-mentioned section 9.3 of GSM standard 03.41, and is illustrated in Table II below:

TABLE II

Sub-Field	Length (bits)	Value
Geographical Scope	2	Indicates a geographical area over which the Message Code is unique, as well as the display mode.
Message Code	10	Varied so as to differentiate between messages from the same source and of the same type (i.e., having same Message Identifier).
Update Number	4	Indicates a change of message content for a given Message Identifier, Geographical Scope and Message Code.

Message Codes are allocated by PLMN operators and can be used to identify different message themes.

Because a given message is not necessarily broadcast by all cells within a given geographical area, the Geographical Scope may be used to determine whether two messages having identical Serial Numbers and Message Identifiers that are received in different cells are indeed identical, as described in the above-mentioned section 9.3 of the GSM 03.41 standard.

Typically, MS 40 operates in a slotted mode, as specified by IS-95B, and does not constantly monitor a paging channel. Therefore, in transmitting a cell broadcast message, special procedures are preferably used to ensure that MSs operating in the slotted mode are able to receive the message.

Thus, in a preferred embodiment of the present invention, BSS 32 transmits a broadcast page message, such as a General Page Message, as specified by IS-95B, or a Paging Request Message, based on GSM standards, or another suitable message type, announcing the impending transmission of a broadcast message. A page record included in the page message (which normally specifies an identification of the MS to which the message is addressed) preferably contains a broadcast address, indicating to MSs receiving the page message when a cell broadcast message is to be transmitted. The actual cell broadcast message is then transmitted once. When one of the MSs decodes the broadcast address in the page record, it can determine whether the broadcast message is a duplicate of a message already received or is not of interest to the subscriber and, if so, will avoid decoding the broadcast message itself.

In another preferred embodiment, BSS 32 uses a periodic broadcast paging cycle, in accordance with the IS-95B specification, so as to reduce overhead required for sending broadcast messages. In this mode of operation, MS 40 monitors the paging channel during a predetermined slot in which broadcast pages or messages are transmitted, so that the pages and messages need be transmitted only once during each periodic broadcast paging cycle. Preferably, the broadcast page is sent in the first slot of a given cycle, which is monitored by the MS, and the broadcast message itself is transmitted later, in accordance with IS-95B.

Preferably, broadcast message scheduling, generally as described in the GSM 03.41 standard, is used in distributing cell broadcast messages to MSs operating in slotted mode. In this case, the broadcast page message comprises a Schedule Message, in which the Message Type field (noted in Table I) inset to '1'. The Schedule Message contains information about a number of consecutive cell broadcast messages that will be transmitted during a Schedule Period, consisting of one or more periodic broadcast cycles, immediately thereafter. The Schedule Message includes a Message Description for each of the messages, along with a cycle number indicating its time position in the Schedule Period. The Message Description contains information from the broadcast address, so that MSs can ascertain the type and source of each scheduled message and whether it has already received the message.

Each Schedule Message includes fields defining a Begin Cycle Number and an End Cycle Number, indicating the length of the Schedule Period. Preferably, a new Schedule Message is transmitted following the last message of the Schedule Period, i.e., in a cycle given by the preceding End Cycle Number + 1. Several Schedule Messages can be broadcast in reference to the same Schedule

Period, and the Begin Cycle Number indicates the first cycle immediately following the last Schedule Message.

When necessary, the BSS may receive instructions to override the schedule published in the Schedule Message, for example, to transmit a new, high-priority SMSCB message. Thereafter, the previous schedule of cell broadcast messages is resumed.

The methods and protocols described hereinabove apply when MS 40 is in communication with CDMA BSS 32. When MS 40 is in communication with CBC 28 via GSM BSS 30, the communications protocols between these elements are in accordance with GSM standards, substantially without modification.

Although preferred embodiments are described hereinabove with reference to specific TDMA- and CDMA-based communications standards, those skilled in the art will appreciate that the methods and principles described hereinabove may also be used in conjunction with other methods of data encoding and signal modulation. The scope of the present invention encompasses not only the complete systems and communications processes described hereinabove, but also various innovative elements of these systems and processes, as well as combinations and sub-combinations thereof. In particular, although preferred embodiments are described hereinabove with reference to hybrid TDMA/CDMA system 20 and MS 40, it will be understood that the methods and apparatus described may equally be used to convey cell broadcast messages to a MS in a CDMA system without TDMA capabilities.

It will thus be appreciated that the preferred embodiments described above are cited by way of example, and the full scope of the invention is limited only by the claims.

WE CLAIM:

CLAIMS

1. In a GSM mobile wireless telecommunications system, a method for broadcasting messages over a CDMA air interface, comprising:

conveying a message to a base station substantially in accordance with a GSM cell broadcast service protocol; and

transmitting the message to a mobile station over the CDMA air interface.

2. A method according to claim 1, wherein transmitting the message comprises transmitting a message substantially in accordance with a CDMA transmission standard.

3. A method according to claim 2, wherein the CDMA transmission standard comprises IS-95B.

4. A method according to claim 2, wherein transmitting the message comprises encapsulating a GSM cell broadcast message in an IS-95 message.

5. A method according to claim 1, wherein transmitting the message comprises transmitting a message over a paging channel.

6. A method according to claim 5, wherein transmitting the message comprises directing the message to one or more specified cells.

7. A method according to claim 5, wherein transmitting the message comprises transmitting a message so as to have a high likelihood of being received by the mobile station while the mobile station is operating in a slotted mode.

8. A method according to claim 7, wherein transmitting the message comprises transmitting a broadcast page followed by transmission of a message.

9. A method according to claim 8, wherein transmitting the broadcast page comprises transmitting a page in a predetermined slot monitored by the mobile station in a periodic broadcast paging cycle.

10. A method according to claim 9, and comprising transmitting a schedule message including information regarding one or more broadcast pages to be transmitted in a schedule period comprising one or more periodic broadcast paging cycles.

11. A method according to claim 1, wherein transmitting the message comprises transmitting a single message multiple times, substantially in accordance with a GSM cell broadcast command.

12. A method according to claim 11, wherein transmitting the single message comprises assigning a broadcast address to the message, such that when the mobile station receives a second message having the same broadcast address as a first, earlier message, the second message is discarded.

13. A method according to claim 11, wherein transmitting the single message multiple times comprises repeating transmission of the message at a repetition frequency determined responsive to the cell broadcast command.

14. A method according to claim 1, wherein conveying the message comprises receiving a message from a cell broadcast center.

15. A method according to claim 1, wherein transmitting the message comprises transmitting a message header including a message identifier field indicative of a characteristic of the message, and wherein the mobile station determines whether to decode or discard the message responsive to the characteristic.

16. Apparatus for broadcasting short messages from a cell broadcast center over a CDMA air interface, comprising a base station subsystem, which receives the messages from the cell broadcast center substantially in accordance with a GSM cell broadcast service protocol and transmits the message to a mobile station over the CDMA air interface.

17. Apparatus according to claim 16, wherein the base station subsystem transmits the message substantially in accordance with a CDMA transmission standard.

18. Apparatus according to claim 17, wherein the CDMA transmission standard comprises IS-95B.

19. Apparatus according to claim 17, wherein the base station encapsulates a GSM cell broadcast message in an IS-95 message.

20. Apparatus according to claim 16, wherein the base station subsystem transmits the message over a paging channel.

21. Apparatus according to claim 16, wherein the base station subsystem directs the message to one or more specified cells.

22. Apparatus according to claim 16, wherein the base station subsystem transmits the message so as to have a high likelihood of being received by the mobile station while the mobile station is operating in a slotted mode.

23. Apparatus according to claim 22, wherein before transmitting the message, the base station subsystem transmits a page in a predetermined slot monitored by the mobile station in a periodic broadcast paging cycle.

24. Apparatus according to claim 16, wherein the base station subsystem transmits a single message multiple times, substantially in accordance with a GSM cell broadcast command.

25. Apparatus according to claim 24, wherein the base station subsystem repeats transmission of the message at a repetition frequency determined responsive to the cell broadcast command.

26. Apparatus according to claim 24, wherein the base station subsystem assigns a broadcast address to the message, such that when the mobile station receives a second message having the same broadcast address as a first, earlier message, the second message is discarded.

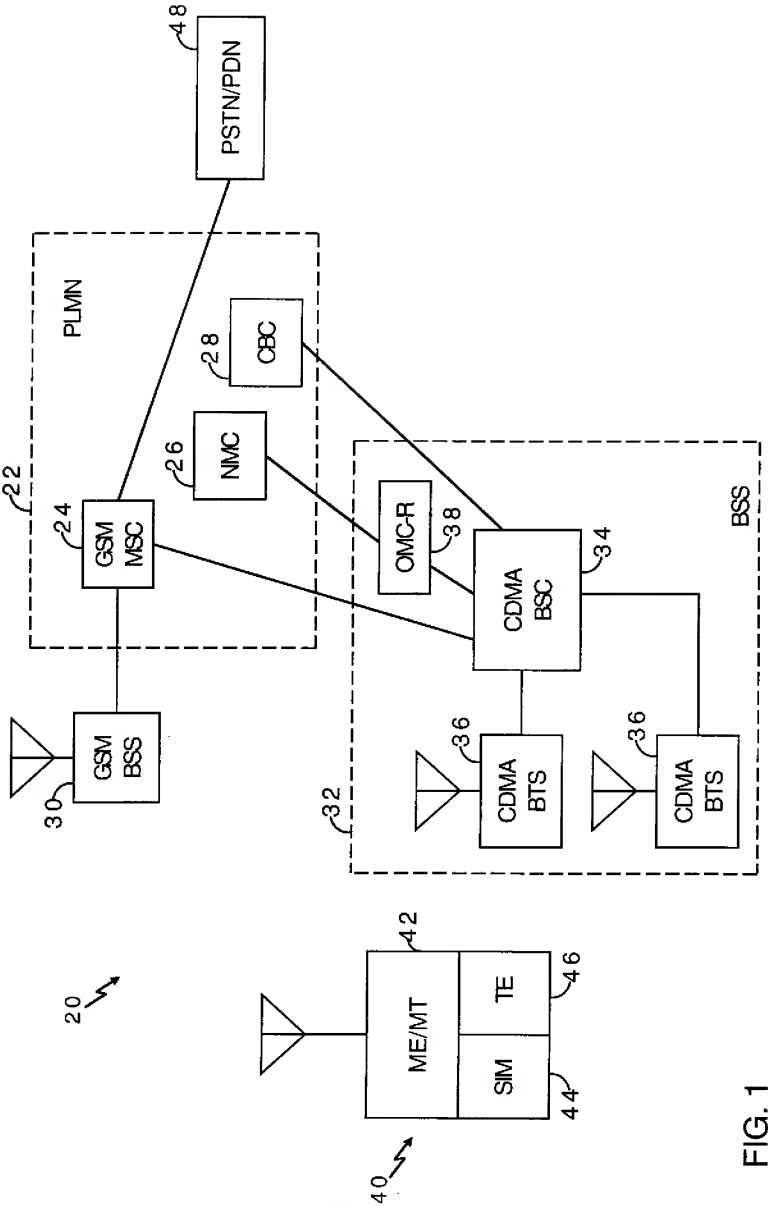

48
PSTN/PDN
PLMN
28
CBC
26
NMC
22
24
GSM MSC
OMC-R
38
34
CDMA BSC
BSS
GSM BSS
30
36
CDMA BTS
36
CDMA BTS
32
20
42
ME/MT
TE
46
SIM
44
40

FIG. 1

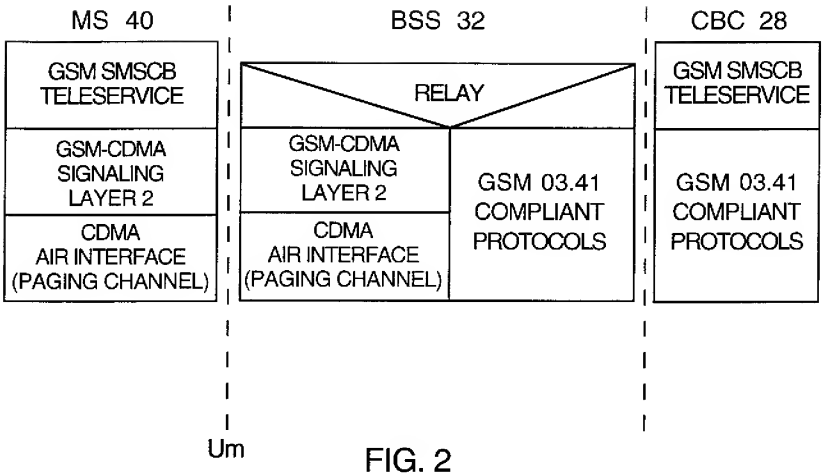

MS 40
BSS 32
CBC 28
GSM SMSCB TELESERVICE
GSM-CDMA SIGNALING LAYER 2
CDMA AIR INTERFACE (PAGING CHANNEL)
RELAY
GSM-CDMA SIGNALING LAYER 2
CDMA AIR INTERFACE (PAGING CHANNEL)
GSM 03.41 COMPLIANT PROTOCOLS
GSM SMSCB TELESERVICE
GSM 03.41 COMPLIANT PROTOCOLS
Um

FIG. 2

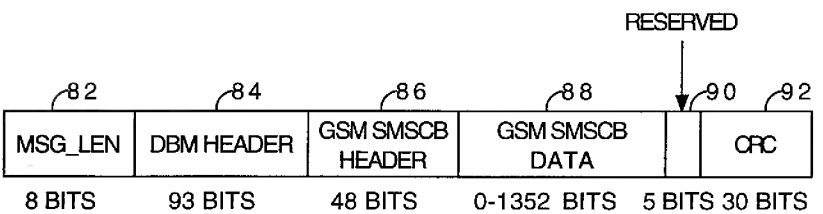

RESERVED
82
84
86
88
90
92
MSG_LEN
DBM HEADER
GSM SMSCB HEADER
GSM SMSCB DATA
CRC
8 BITS
93 BITS
48 BITS
0-1352 BITS
5 BITS
30 BITS

FIG. 3

INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/US 00/21065

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04Q7/22

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04Q

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 920 822 A (BOUDREAU ALAIN ET AL) 6 July 1999 (1999-07-06) column 1, line 66 -column 2, line 11 column 2, line 49 -column 3, line 22 column 4, line 19 - line 37 column 8, line 9 - line 23 ---	1-26
A	FENTON C J ET AL: "MOBILE DATA SERVICES" BT TECHNOLOGY JOURNAL,GB,BT LABORATORIES, vol. 14, no. 3, 1 July 1996 (1996-07-01), pages 92-108, XP000598159 ISSN: 1358-3948 page 101, paragraph 4.2 ---	1-26

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
6 November 2000	14/11/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Rothlübbers, C

1

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE: "DIGITAL CELLULAR TELECOMMUNICATIONS SYSTEM (PHASE 2+);TECHNICAL REALIZATION OF SHORT MESSAGE SERVICE CELL BROADCAST (SMSCB) (GSM 03.41 VERSION 5.8.1)" EUROPEAN TELECOMMUNICATION STANDARD,XX,XX, no. ETS 300 902, June 1998 (1998-06), pages 1-30, XP002128897 -----	

1